
# Creación de plugins para el paquete R Commander de R y aplicaciones

Tesis de Máster
Máster en Matemática Computacional
Departamento de Matemáticas
Universitat Jaume I

**Javier Osmar Artola García**

Director: Pablo Gregori Huerta

**Castellón de la Plana, 16 de noviembre de 2015**

Certifico que el presente trabajo ha sido realizado bajo mi supervisión por el alumno Javier Osmar Artola García, y se presenta para que constituya el Trabajo de Fin de Máster del Máster de Matemática Computacional.

Firmado: Pablo Gregori Huerta

# Índice general

**Índice de figuras** III

**Índice de tablas** IV

**Resumen** V

**1. Introducción a R** 1
1.1. ¿Qué es R? . . . 1
1.2. Historia y evolución de R . . . 2
1.2.1. Breve historia de R . . . 2
1.2.2. Evolución de R . . . 3
1.3. Interfaz gráfica de R (R Gui) . . . 4
1.4. Estructura de R . . . 6
1.5. El lenguaje de programación R . . . 7
1.5.1. Tipos de datos . . . 7
1.5.2. Operadores . . . 8
1.5.3. Variables - Vectores . . . 8
1.5.3.1. Vectores enteros . . . 8
1.5.3.2. Vectores numéricos . . . 11
1.5.3.3. Vectores de caracteres . . . 11
1.5.3.4. Vectores lógicos . . . 12
1.5.4. Valores faltantes . . . 12
1.5.5. Factores . . . 13
1.5.6. Matrices . . . 15
1.5.7. Listas . . . 18
1.5.8. Hojas de datos . . . 20
1.5.9. Control de flujo . . . 21
1.5.9.1. El ciclo `for()` . . . 21
1.5.9.2. El comando `if()` . . . 21
1.5.9.3. El comando `repeat` . . . 22
1.5.9.4. El ciclo `while()` . . . 22

1.5.10. Funciones . . . . . 22
1.5.11. Funciones estadísticas . . . . . 23
1.5.11.1. Intervalo de confianza . . . . . 23

**2. El paquete R Commander** **26**
2.1. Aspectos básicos de R Commander . . . . . 26
2.2. Introducir datos . . . . . 30
2.3. Resúmenes numéricos . . . . . 30
2.4. Crear Gráficas . . . . . 31
2.5. Modelos estadísticos . . . . . 32
2.6. Salir de Rcmdr . . . . . 33
2.7. Paquetes Plug - In . . . . . 33

**3. Extensión de R Commander** **36**
3.1. Definición del archivo menú . . . . . 36
3.2. Construyendo un nuevo menú . . . . . 39
3.3. Añadir una función a R Commander . . . . . 39
3.4. Creación de paquetes Plug-In para R Commander . . . . . 43
3.4.1. Creación de un paquete de R en Windows . . . . . 44
3.4.2. Creación de nuevas opciones de menú por plug-ins de R Commander . . . . . 45

**4. Menú para cálculo de probabilidades** **47**
4.1. Distribuciones de probabilidad . . . . . 47
4.2. Distribución binomial . . . . . 48
4.3. Nuestro propósito . . . . . 49
4.4. El plug-in `RcmdrPlugin.Artola` . . . . . 51
4.5. Conclusiones . . . . . 52

**Bibliografía** **53**

**A. Anexo** **55**

# Índice de figuras

1.1. Pantalla inicial de R . . . . . 5

2.1. Pantalla de Rcmdr . . . . . 27
2.2. Cuadro de diálogo del modelo lineal . . . . . 32
2.3. Menús de RcmdrPlugin.BCA . . . . . 35

3.1. Archivo Rcmdr-menus . . . . . 37
3.2. Menú Artola . . . . . 39
3.3. Editando el archivo Rcmdr-menus.txt . . . . . 40
3.4. Menú agregado a Rcmdr . . . . . 41
3.5. Ventana de diálogo para la función Box-Cox . . . . . 43
3.6. Ventana de diálogo para la función Confint . . . . . 43

4.1. Menú de la distribución Binomial . . . . . 48
4.2. Probabilidades binomiales acumuladas . . . . . 48
4.3. Probabilidades binomiales . . . . . 49

# Índice de tablas

1.1. Funciones estadísticas . . . . . 24
1.2. Definiciones alternativas de algunas funciones de (densidad) de probabilidad, a modo de ejemplo. . . . . 25

# Resumen

El presente trabajo consiste en la creación de paquetes de tipo plugin, para añadir menús al paquete R Commander de R.

En el primer capítulo se describe una breve introducción, aspectos históricos, evolución, interfaz gráfica, estructura, los tipos de datos, operadores, variables, descripción del lenguaje de programación y funciones del software R.

El segundo capítulo refiere al paquete R Commander de R, en el que se abordan los aspectos teóricos, la introducción de datos, crear resúmenes numéricos, construcción de gráficos, ajuste de modelos estadísticos y se ha dado una breve descripción de aspectos teóricos de los paquetes plugins.

En el capítulo tres se muestra cómo añadir menús al paquete R Commander, esto lo podemos lograr a partir de dos formas, la primera es modificar el código fuente del paquete R Commander y reconstruirlo, la segunda es a partir de la creación de archivos con extensión `.R`. Aquí se han añadido menús a partir de paquetes nuevos, sin modificar el código fuente del paquete R Commander.

El capítulo cuatro estuvo dedicado a crear un menú más natural al que muestra R Commander para calcular probabilidades en modelos de variables aleatorias discretas y continuas.

Para lograr con éxito lo antes descrito, lo primero que se tuvo que hacer es trazar una imagen clara de la forma que debía tener el nuevo menú, cabe aclarar que éste puede variar con el gusto de cada usuario, luego se procedió a programar en R las funciones que permiten realizar el cálculo de probabilidades.

Varias de las funciones admiten parámetros iniciales, por tanto, se crearon en lenguaje `tcl-tk` las ventanas de diálogos con sus respectivas etiquetas, botones y entradas.

Palabras clave: código fuente, lenguaje R, lenguaje tcl-tk, menú del paquete R Commander.

# Capítulo 1

# Introducción a R

## 1.1. ¿Qué es R?

R es un potente software estadístico en el cual podemos trabajar muchas herramientas estadísticas, por mencionar algunas: modelos lineales y no lineales, test estadísticos, análisis de series temporales, control estadístico de la calidad, algoritmos de clasificación y agrupamiento, etc. y gráficas.

En [17], encontramos que R es un conjunto integrado de programas para manipulación de datos, cálculo y gráficos. Entre otras características dispone de:

- almacenamiento y manipulación efectiva de datos,
- operadores para cálculo sobre variables idexas (arrays), en particular matrices,
- una amplia, coherente e integrada colección de herramientas para análisis de datos,
- posibilidades gráficas para análisis de datos, que funcionan directamente sobre pantalla o impresora, y
- un lenguaje de programación bien desarrollado, simple y efectivo, que incluye condicionales, ciclos, funciones recursivas y posibilidad de entradas y salidas (debe destacarse que muchas de las funciones suministradas con el sistema están descritas en el lenguaje R).

La definición anterior [17] establece que R no sólo se utiliza en el área de la estadística, sino que es bastante usado en diversas ramas del conociento científico, tales como: la investigación biomédica, biología, sismología, psicología, programación lineal, física, ecuaciones diferenciales, matemáticas financieras, etc.

La ventaja principal es que es de caracter libre bajo licencia GNU General Public Licence (Licencia Pública General) y lo podemos descargar en su sitio oficial `http://www.r-project.org` para cualquiera de los sistemas operativos Windows, (Mac) OS X, iOS, Linux y otros.

Debido a que existe para diversidad de sistemas operativos se denomina multiplataforma. De hecho, esta es otra de las ventajas que nos facilita a los usuarios.

La información en la página oficial de R (citada arriba), está estructurada de la siguiente manera:

- **Download:** Entramos a descargar R; lo primero que debemos hacer es seleccionar un CRAN (Comprehensive R Archive Network), debemos elegir el más cercano a nuestra localización o el país que tenga nuestro mismo idioma, luego seleccionamos el sistema operativo bajo el cual instalaremos R.
- **R project:** Aquí encontramos información acerca de R, los colaboradores, lo nuevo de R, listas de correos para tener ayuda por temas (anuncios sobre R, desarrollo de paquetes, paquetes, etc.), seguimiento a fallos, conferencias y busqueda.
- **R Foundation:** Encontramos una breve descripción sobre la fundación R, junta directiva y auditores, miembros de la fundación, donantes y donaciones por si alguien quiere apoyar monetariamente.
- **Documentation:** Encontramos manuales, FAQs (preguntas frecuentes), revista de R, libros, certificación y otra documentación.
- **Links:** Encontramos información y direcciones web sobre proyectos relacionados con R y bioconductor.

## 1.2. Historia y evolución de R

Acá se abordará de manera resumida aspectos históricos de R y la evolución que éste ha tenido a través del tiempo.

### 1.2.1. Breve historia de R

Fue desarrollado en el año 1992 por Robert Gentleman y Ross Ihaka del departamento de Estadística de la universidad de Auckland en Nueva Zelanda, cabe mencionar que el primer anuncio del software salió al público en 1993 y se nombró R debido a las iniciales de los nombres de los creadores antes mencionados. En la actualidad R se mantiene mediante la contribución de muchos científicos de todo el mundo que hacen uso de éste e incorporan mejoras para brindar un mejor

funcionamiento al resto de usuarios, su desarrollo actual es responsabilidad del R Development Core Team.

R puede considerarse como una implementación del lenguaje S, veremos a continuación una breve historia de S para entender un poco más sobre los fundamentos de R.

El lenguaje S fue desarrollado por John Chambers y colaboradores en Laboratorios Bell (AT&T), actualmente Lucent Tecnologies, en 1976. Este lenguaje, originalmente fue codificado e implementado como unas bibliotecas de FORTRAN. Por razones de eficiencia, en 1988 S fue reescrito en lenguaje C, dando origen al sistema estadístico S, Versión 3. Con la finalidad de impulsar comercialmente a S, Bell Laboratories dio a StatSci (ahora Insightful Corporation) en 1993, una licencia exclusiva para desarrollar y vender el lenguaje S. En 1998, S ganó el premio de la Association for Computing Machinery a los sistemas de software, y se liberó la versión 4, la cual es prácticamente la versión actual [16].

El éxito de S fue tal que, en 2004 Insightful decide comprar el lenguaje a Lucent (Bell Laboratories) por la suma de 2 millones de dólares, convirtiéndose hasta la fecha en el dueño. Desde entonces, Insightful vende su implementación del lenguaje S bajo el nombre de S-PLUS, donde le añade un ambiente gráfico amigable. En el año 2008, TIBCO compra Insightful por 25 millones de dólares y se continúa vendiendo S-PLUS, sin modificaciones. R, que define su sintaxis a partir de esa versión de S, no ha sufrido en lo fundamental ningún cambio dramático desde 1998 [16].

### 1.2.2. Evolución de R

En el año 1995 Martin Mächler, de la Escuela Politécnica Federal de Zúrich, convence a Ross y Robert a usar la licencia GNU para hacer de R un software libre [16].

Con el propósito de crear algún tipo de soporte para el lenguaje, en 1996 se crea una lista pública de correos; sin embargo debido al gran éxito de R, los creadores fueron rebasados por la continua llegada de correos. Por esta razón, se vieron en la necesidad de crear, en 1997, dos listas de correos, a saber: R-help y R-devel, que son las que actualmente funcionan para responder las diversas dudas que los usuarios proponen en muy diversos asuntos relativos al lenguaje [16].

R ha experimentado un aumento en la cantidad de paquetes o librerías [1], un ejemplo claro de esto es que en la versión 3.3.0 (2015-07-04) junto a R se incluyen 25 bibliotecas (llamadas bibliotecas estándar), en tanto la versión 1.0.1 (2000-05-16) contenía apenas 8 de éstas.

[1] Un paquete es es una colección de funciones y programas usados internamente por R [1].

Entrando a paquetes en `http://ftp.cixug.es/CRAN/`, encontramos que el repositorio de paquetes CRAN cuenta con 6,964 paquetes disponibles, vemos también, una tabla de paquetes disponibles ordenados por fecha de publicación, además se muestra una tabla en la que aparecen los paquetes ordenados por nombre.

Adicional a lo descrito anteriormente, existen muchos paquetes más que han desarrollados ciertos usuarios y otros que los podemos encontrar en redes personales.

Para instalar paquetes tenemos dos opciones, la primera es directamente desde la consola tecleando el comando:

```
> install.packages("Nombre del paquete")
```

La otra opción es desde la barra de menú:

`Paquetes` → `Instalar paquetes(s)...`

Dado el enorme número de nuevos paquetes, éstos se han organizado en vistas (o temas), que permiten agruparlos según su naturaleza y función. Por ejemplo, hay grupos de paquetes relacionados con estadística bayesiana, econometría, series temporales, etc.

Otro avance de R es que sus creadores y colaboadores están pendientes de incluir librerías para el desarrollo de temáticas nuevas, por ejemplo las versiones actuales contienen extensiones específicas para áreas como la bioinformática, geoestadística y modelos gráficos.

## 1.3. Interfaz gráfica de R (R Gui)

La primera ventana que vemos al iniciar una sesión en R (seamos nuevo o viejo usuario) se muestra en la Figura 1.1, esta es la denominada consola (R console). El simbolo (en rojo) "`>`" (prompt) nos indica que R está esperando la entrada de órdenes, las cuales escribimos a continuación de éste.

Las órdenes son separadas mediante punto y coma ("`;`") o mediante un cambio de línea. Si se escribe un código incompleto el programa presenta un "`+`" que nos indica que debemos completar la orden.

Una ventaja es que podemos personalizar la consola con un tipo de letra, tamaño, color, etc., esto lo hacemos desde el menú Editar, Preferencias de la interfaz gráfica. Hecho lo antes descrito, al cerrar sesión, R nos mandará un rótulo diciendo: Guardar imagen de área de trabajo?, damos Sí y cuando iniciemos una nueva sesión lo podemos hacer desde acá y veremos nuestra interfaz personalizada.

Para cerrar el programa podemos hacerlo directamente desde la consola tecleando la orden `q()`. Si estamos trabajando con datos nos preguntará si queremos

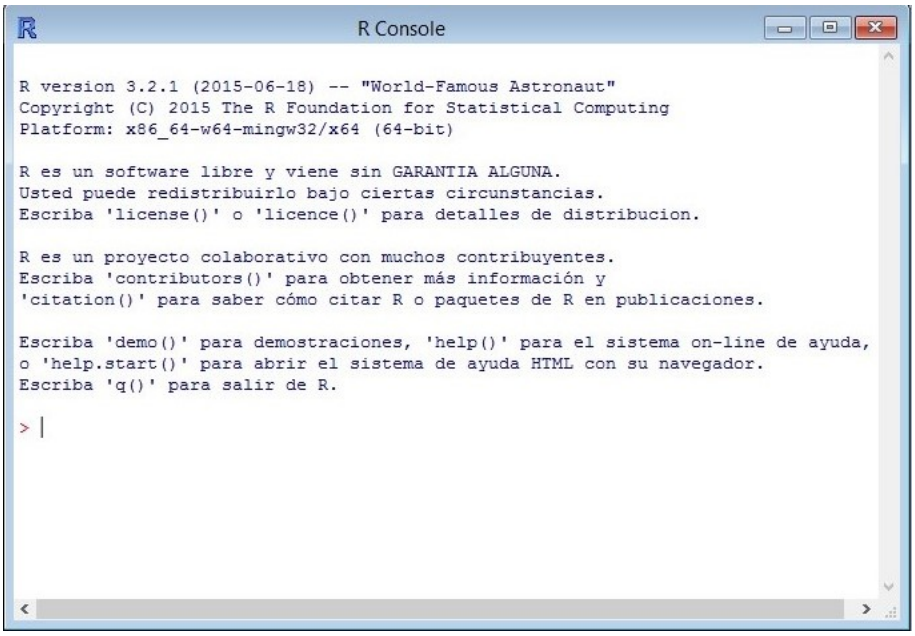


Figura 1.1: Pantalla inicial de R

guardar la sesión de trabajo. Si guardamos, los datos estarán disponibles en la siguiente sesión de R.

El trabajo en R se hace mediante líneas de comandos que se escriben en la consola. Esto se vuelve un poco tedioso ya que sólo podemos ejecutar al mismo tiempo una línea de código. Esto significa que esta forma de trabajo es útil únicamente para ejecutar acciones simples.

Para ejecutar varias líneas de código, se debe crear un archivo de texto con las mismas, se lanza a la consola y ejecuta con el comando `source` ("Nombre del archivo"). La consola de R puede facilitar ese proceso: permite, desde su menú `Archivo`, crear nuevo script. Entonces se abre la ventana R Editor donde se editan las líneas de comandos, y desde su menú `Editar`, se puede "Correr línea o selección" o "Ejecutar todo", lanzándose el código a la consola y ejecutándose. La ventana R Editor tiene muy pocas funcionalidades, por lo que se recomienda otras interfaces para un trabajo intensivo con R.

Para hacer más amigable nuestro trabajo en R, existen varias GUIs (Graphical User Interface) [2], algunas de éstas nos permiten trabajar sin hacer uso de comandos, ya que tienen disponibles barras de menús y botones, entre ellas tenemos:

- RKWard (`http://rkward.sourceforge.net/`)

[2]Interfaz gráfica de usuario

- RStudio (`http://www.rstudio.com/`)
- Rweka (`http://www.cs.waikato.ac.nz/~ml/weka/index.html`)
- R Commander
- REXCEL(`http://www.statconn.com/products.html`)
- Poor Man's GUI (`http://www.math.csi.cuny.edu/pmg`)
- R.NET (`http://www.u.arizona.edu/~ryckman/RNet.php`)

## 1.4. Estructura de R

En general R está estructurado por una serie de paquetes modulares los cuales pueden ser descargados e instalados en R para su respectivo uso, cabe aclarar que, cada paquete es útil únicamente para lo que fue creado, es por tal razón que existen muchos de éstos.

Cada uno de los paquetes tiene una estructura de funcionalidad que podemos encontrar en libros, artículos, revistas, páginas web, etc. Además desde la ayuda en R hallamos toda la información relacionada a cada paquete, en varios casos nos muestran ejemplos de como funciona cada uno.

El sistema base de R contiene el paquete básico que se requiere para su ejecución y la mayoría de las funciones fundamentales. Los otros paquetes contenidos en la "base" del sistema incluye a **utils, stats, datasets, graphics, grDevices, grid, tools, parallel, compiler, splines, tcltk, stats4** [16].

R posee una excelente capacidad gráfica, que permite generar gráficos de alta calidad. Según [17], las órdenes gráficas se dividen en tres grupos básicos:

- **Alto nivel** son funciones que crean un nuevo gráfico, posiblemente con ejes, etiquetas, títulos, etc..
- **Bajo nivel** Son funciones que añaden información a un gráfico existente, tales como puntos adicionales, líneas y etiquetas.
- **Interactivas** Son funciones que permiten interactuar con un gráfico, añadiendo o eliminando información, utilizando un dispositivo apuntador, como un ratón.

Es importante mencionar que, debido a su estructura, R consume mucho recurso de memoria, por lo tanto si se utilizan datos de tamaño enorme, el programa se ralentizaría o, en el peor de los casos, no podría procesarlos. En la mayoría de los casos, sin embargo, los problemas que pudieran surgir con referencia a la

lentitud en la ejecución del código, tienen solución, principalmente teniendo cuidado de vectorizar el código; ya que esto permitiría particionarlo y aprovechar en procesamiento paralelo en equipos con multi-nucleos [16].

## 1.5. El lenguaje de programación R

El lenguaje de programación R es un lenguaje más, con características similares a los lenguajes orientados a objetos (como el C++). Está diseñado especialmente para el cálculo estadístico, por lo que la clase más básica es el *vector*, concepto ideal para almacenar una muestra de datos.

Es un lenguaje interpretado, y no compilado, por lo que su velocidad ejecución no puede competir con otros lenguajes. Los paquetes que extienden las funcionalidades de R suelen incluir funciones escritas y compiladas en C, Fortran u otros lenguajes, y el lenguaje R queda para cálculos que no necesitan optimizar la velocidad.

Sin embargo, el lenguaje R queda como un lenguaje puente. Aunque es código abierto, y cualquiera lo puede examinar, las funciones ya compiladas quedan como "caja negra" para el usuario que no domina el lenguaje en el que fueron compiladas. El usuario, gracias a la ayuda proporcionada por el desarrollador del paquete, sabe los argumentos que debe pasar, y los objetos que devuelve la función. Entonces R permite manipular dichos objetos retornados (hacer transformaciones con operadores matemáticos u otras funciones, presentar tablas, gráficos, etc.), y habitualmente, llamar a otras funciones compiladas que continúan el análisis estadístico deseado, pasando como argumentos dichos objetos.

### 1.5.1. Tipos de datos

En R un dato puede ser almacenado bajo diferentes formas, existen varios tipos de datos, los más comunes se muestran a continuación.

- **Numeric:** Es cualquier número decimal. Para separar decimales se utiliza el punto, cualquier número que tecleemos por defecto tomará este tipo. Si queremos saber si un dato es de tipo numérico podemos usar la orden `is.numeric()`.

- **Integer:** Cualquier número entero. Si queremos convertir a integer un número que declaramos inicialmente como tipo Numeric, se utiliza el comando `as.integer()`.

- **Complex:** Son los números complejos, no se usan en análisis de datos.

- **Logical:** En este caso puede tomar cualquiera de los valores lógicos TRUE (verdadero) o FALSE (falso).
- **Character:** Es cualquier cadena de caracteres alfanuméricos y debe introducirse entre comillas. Para convertir cualquier número en una cadena de caracteres se usa el comando `as.character()`.

### 1.5.2. Operadores

Los valores de cada tipo de datos pueden efectuarse haciendo uso de distintos operadores o funciones predefinidas para cada uno de éstos.

Existe un orden de prioridad al momento de evalúar expresiones aritméticas, así que se evalúan en el siguiente orden: las funciones predefinidas, las potencias, los productos y cocientes, las sumas y restas, operadores de comparación, la negaciones, las conjunciones y las disyunciones.

Si deseamos un orden de evalución distinto al anterior (predefinido) debemos hacer uso de paréntesis. A continuación se muestran los operadores más usuales.

- Operadores aritméticos: $+$, $-$, $*$, $/$ y $\hat{}$
- Operadores de comparación: $>$, $<$, $>=$, $<=$, $==$ y $!=$
- Operadores lógicos: Conjunción &, disyunción |, negación !

### 1.5.3. Variables - Vectores

Los datos en R son almacenados como vectores, esto significa que las variables son representadas por vectores. Un vector es una colección de una o más entradas de la misma clase.

Los vectores son un arreglo unidimensional de entradas, esto es, una única fila (o columna) de entradas, cuya longitud es el número de columnas en la fila o viceversa.

Cada entrada tiene un único número indexo que es equivalente al número de la fila que puede ser usada para referirse a una entrada (interna al vector) en particular.

A continuación tenemos las cuatro clases de vectores más importantes en R.

#### 1.5.3.1. Vectores enteros

Vector de números con elementos enteros. Con este tipo de vectores se definen las variables[3] cuantitativas discretas de la estadística. Supongamos que queremos

[3]Para definir variables en R, debemos ser cuidadosos con los nombres que usemos ya que distingue entre mayúsculas y minúsculas.

definir la variable número de hermanos, para esto hacemos:

```
> nh=c(2,1,3,1,4,3,2,5,2,3,3,3,4,5,1,2,2,2)
```

La asignación se realiza mediante el comando "=", la función `c()` se encarga de crear un vector con los argumentos, que en este caso son constantes numéricas.

Cuando se definen vectores con longitud extensa y no la podemos ver a simple vista, hacemos uso de la función `length()` para conocer su longitud (cantidad de elementos). Para la variable anterior, tenemos:

```
> nh=c(2,1,3,1,4,3,2,5,2,3,3,3,4,5,1,2,2,2)
> length(nh)
[1] 18
```

Una tercera forma de introducir los datos es crear un conjunto de datos (hoja de datos) mediante el comando `data.frame()`. Por ejemplo, supongamos que queremos crear un conjunto de datos al que llamaremos `familia`, a partir de la variables edad, sexo, lugar de origen y etnia, así:

```
edad=c(28,30,25,27,29)
sexo=c("Femenino","Masculino","Femenino","Femenino","Masculino")
lugarorigen=c("Rosita","Siuna","Siuna","Rosita","Rosita")
etnia=c("Mestizo","Mestizo","Mestizo","Mestizo","Mestizo")
familia=data.frame(edad,sexo,lugarorigen,etnia)
familia
   edad      sexo lugarorigen   etnia
1   28  Femenino      Rosita Mestizo
2   30 Masculino       Siuna Mestizo
3   25  Femenino       Siuna Mestizo
4   27  Femenino      Rosita Mestizo
5   29 Masculino      Rosita Mestizo
```

Hemos creado una matriz de datos en la que cada columna se corresponde con una variable y cada fila con un individuo. En nuestro caso la matriz está compuesta por cinco columnas que corresponden a edad, sexo,lugar de origen y etnia y 5 filas, una para cada individuo.

Si queremos acceder a las variables de un conjunto de datos, usamos el operador de dolar $, en nuestro caso para acceder a `edad`, tecleamos:

```
> familia$edad
[1] 28 30 25 27 29
```

Para añadir nuevas variables a un conjunto de datos, debemos tener cuidado de que tengan el mismo tamaño muestral, así supongamos que queremos añadir una nueva variable con el nombre de cada individuo, hacemos:

```
familia$nombre=c("Julissa","Osmar","Sayra","Nelly","Orlando")
```

Hemos agregado una nueva columna al conjunto de datos (llamdo `familia`) tendrá, veamos a continuación.

```
familia
```

```
  edad      sexo lugarorigen   etnia  nombre
1   28  Femenino      Rosita Mestizo Julissa
2   30 Masculino       Siuna Mestizo   Osmar
3   25  Femenino       Siuna Mestizo   Sayra
4   27  Femenino      Rosita Mestizo   Nelly
5   29 Masculino      Rosita Mestizo Orlando
```

Supongamos que queremos obtener la el nombre y edad de la tercera persona, hacemos:

```
print(familia$nombre[3]);print(familia$edad[3])
[1] "Sayra"
[1] 25
```

Las variables que definimos en cada sesión de trabajo quedan almacenadas en la memoria interna de R en lo que se conoce como espacio de trabajo.

Podemos obtener un listado de todos los objetos almacenados en el espacio de trabajo mediante la función `ls()`, si queremos más información como los objetos de la memoria, sus tipos y sus valores, usamos `ls.str()`.

```
ls()
```

```
[1] "edad"  "etnia"  "familia"  "lugarorigen"  "sexo"
```

```
ls.str()
```

```
edad: num [1:5] 28 30 25 27 29
etnia: chr [1:5] "Mestizo" "Mestizo" "Mestizo" "Mestizo" "Mestizo"
familia: 'data.frame': 5 obs. of  5 variables:
 $ edad: num  28 30 25 27 29
 $ sexo: Factor w/ 2 levels "Femenino","Masculino": 1 2 1 1 2
```

```
 $ lugarorigen: Factor w/ 2 levels "Rosita","Siuna": 1 2 2 1 1
 $ etnia: Factor w/ 1 level "Mestizo": 1 1 1 1 1
 $ nombre: chr "Julissa" "Osmar" "Sayra" "Nelly" ...
lugarorigen: chr [1:5] "Rosita" "Siuna" "Siuna" "Rosita" "Rosita"
sexo: chr [1:5] "Femenino" "Masculino" "Femenino" "Femenino" ...
```

Para eliminar objetos de la memoria se usa el comando `rm()`, su sintaxis es la siguiente:

```
rm(objeto1, objeto2, ...)
```

Veamos a continuación como eliminar la variables etnia de nuestro conjunto de datos (`familia`).

```
ls()
```

```
[1] "edad"  "etnia"  "familia"  "lugarorigen"  "sexo"
```

```
rm(etnia)
ls()
```

```
[1] "edad"  "familia"  "lugarorigen"  "sexo"
```

#### 1.5.3.2. Vectores numéricos

Vectores cuyos elementos son números reales. Por ejemplo definamos el vector para la variable `peso` (en kg) que fue tomada en cinco estudiantes.

```
> peso=c(64.5,63.2,68.7,59.2,66.9)
```

Los vectores de tipo numéricos son usados para definir a las variables cuantitativas continuas.

#### 1.5.3.3. Vectores de caracteres

Cadena de caracteres o letras. Para construirlos escribimos entre comillas la sucesión de caracteres que los definen. Esta clase de vectores pueden concatenarse en uno sólo mediante la función `c()`. Veamos el siguiente ejemplo.

```
x=c("Julissa","Nathjuly","Orlandito","Sergito","Luis Neil")
```

Las variables categóricas son representadas por vectores de caracteres.

#### 1.5.3.4. Vectores lógicos

Vector cuyos elementos pueden tomar dos valores TRUE (verdadero) o FALSE (falso). Este tipo de vectores son muy útilies cuando usamos condiciones.

Por ejemplo, si queremos que la variable `edad` cumpla la condición de ser menor a 25 años, hacemos:

```
edad<25
```

```
[1] FALSE FALSE FALSE FALSE FALSE
```

Veamos que la salida toma los valores `TRUE` o `FALSE` de acuerdo a que los elementos de `edad` cumplan o no la condición especificada, en nuestro caso la variable `edad` sólo toma valores `FALSE` ya que ninguna de las edades (del conjunto de datos `familia`) son menores a 25 años

Muchas veces los vectores lógicos son usados en expresiones aritméticas; pero primero se deben convertir a vectores numéricos usando codificación binaria, esto es, `FALSE` se transforma en 0 y `TRUE` en 1. Cabe mencionar que existen casos en los cuales el vector lógico y su correspondiente numérico no son equivalentes, como veremos a continuación.

### 1.5.4. Valores faltantes

Cuando no conocemos al menos una componente de un vector, decimos que tenemos valores faltantes y los representamos por `NA`[4]. Para indicar valores faltantes usamos la función genérica `is.na()`, la cual toma los valores TRUE en caso de que declaremos algún valor faltante y FALSE en caso contrario.

A manera de ejemplo definamos el vector "`a`" sin valores faltantes.

```
> a=c(21,22,23,24,22,34,44,22,33)
> is.na(a)
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Notemos que todos los valores que tomó la función `is.na()` son FALSE, esto se debe a que no hemos definido algún valor faltante.

Ahora veamos otro ejemplo en el cual definimos dos valores faltantes, esto es:

```
j=c(21,22,12,15,NA,NA)
> is.na(j)
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE
```

[4] "Not Available/Missing Values", No disponible/Valores perdidos

Los dos últimos valores de la función `is.na()` son TRUE, esto era de esperar debido a que en el vector `j` los declaramos como faltantes.

En procesos de cálculos se produce una segunda forma de valores faltantes, estos son los valores `NaN` [5].

R no realiza cálculo simbólico, pero tiene la constante `Inf` que se puede utilizar directamente u obtener tras operaciones como 1/0. Los valores `NaN` se obtienen cuando realizamos operaciones como las siguientes:

```
> Inf/Inf
[1] NaN
> 0/0
[1] NaN
> Inf-Inf
[1] NaN
```

Para investigar si una expresión numérica es de tipo `NaN`, usamos la función `is.nan()`, veamos unos ejemplos.

```
> is.nan(0/0)
[1] TRUE
> is.nan(1/0)
[1] FALSE
```

Vea que las salidas de la función `is.nan()` serán TRUE en caso de que la operación sea de tipo `NaN` y FALSE en caso contrario.

### 1.5.5. Factores

Un factor es un vector que se utiliza para clasificar de manera discreta los elementos de otro vector de igual longitud.

Supongamos que queremos conocer la frecuencia del lugar de origen de ciertas personas, para esto creamos la siguiente variable.

```
> lugar.origen=c("Siuna","Rosita","Rosita","Rosita","Siuna",
"Managua","Managua","Chinandega")
```

Para transformar la variable `lugar.origen` a un factor, debemos hacer uso de la función de conversión `as.factor()`, esto es:

[5] "Not a Number", No es un número

```
> Flugar.origen=as.factor(lugar.origen)
> Flugar.origen
[1] Siuna      Rosita     Rosita     Rosita     Siuna
Managua     Managua
[8] Chinandega
Levels: Chinandega Managua Rosita Siuna
```

La función `levels()` nos proporciona los niveles de un factor, en nuestro caso:

```
> levels(Flugar.origen)
[1] "Chinandega" "Managua"    "Rosita"     "Siuna"
```

La frecuencia de aparición de ciertos elementos en un vector pueden obtenerse mediante la función `table()` que toma típicamente como argumento un factor y regresa como resultado justamente la frecuencia de apación de los niveles en el vector de índices.

```
> table(Flugar.origen)
Flugar.origen
Chinandega    Managua     Rosita      Siuna
         1           2          3          2
```

Para acceder a un elemento individual de un factor, hacemos:

```
> Flugar.origen[4]
[1] Rosita
Levels: Chinandega Managua Rosita Siuna
```

Ahora, para acceder a un elemento individual de los niveles.

```
> levels(Flugar.origen)[2]
[1] "Managua"
```

Supongamos que tenemos en otro vector los ingresos (en córdobas) de las personas a las cuales hemos recolectado su lugar de origen.

```
> ingresos=c(8000,12000,15000,7000,6000,13000,7500,7900)
```

Para calcular la media muestral para cada lugar de origen, usamos la función `tapply()` que da como resultado el vector de medias con las componentes etiquetadas con los niveles.

```
> MediaIngresos=tapply(ingresos,Flugar.origen,mean)
> MediaIngresos
Chinandega    Managua     Rosita      Siuna
   7900.00   10250.00   11333.33    7000.00
```

### 1.5.6. Matrices

Desde el punto de vista del lenguaje, una matriz es un vector con un atributo adicional: `dim`. Para el caso de las matrices, este atributo es un vector entero de dos elementos, a saber: el número de renglones y el número de columnas que componen la matriz [16].

Para crear matrices usamos la función `matrix()`. Veamos un ejemplo de matriz.

```
> j=matrix(c(2,6,8,9),nrow=2)
> j
     [,1] [,2]
[1,]    2    8
[2,]    6    9
```

Hemos creado una matriz de orden dos. Una forma alternativa de crear matrices es:

```
> a=matrix(c(2,1,3,2,4,1),2,3)
> a
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    1    2    1
```

Si queremos obtener un elemento de la matriz, por ejemplo el elemento de la segunda fila y la segunda columna, hacemos:

```
> a[2,2]
[1] 2
```

Podemos hacer referencia a una fila o una columna de la matriz como si se tratara de un sólo objeto, esto es como un vector. Dada la siguiente matriz:

```
> k=matrix(c(2,0,0,0,0,7,-3,0,0,0,-3,7,6,0,0,8,5,7,9,0,3,1,6,8,4),
5,5)
> k
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    7   -3    8    3
[2,]    0   -3    7    5    1
[3,]    0    0    6    7    6
[4,]    0    0    0    9    8
[5,]    0    0    0    0    4
```

Extraemos la fila 3 y la columna 4, así:

```
> k[3,]
[1] 0 0 6 7 6

> k[,4]
[1] 8 5 7 9 0
```

A las filas y a las columnas de una matriz se les puede asignar nombres, esto es:

```
> rownames(k)=c("Uno","Dos","Tres","Cuatro","Cinco")
> colnames(k)=c("Uno","Dos","Tres","Cuatro","Cinco")
> k
       Uno Dos Tres Cuatro Cinco
Uno      2   7   -3      8     3
Dos      0  -3    7      5     1
Tres     0   0    6      7     6
Cuatro   0   0    0      9     8
Cinco    0   0    0      0     4
```

Para extraer un elemento individual, a partir de los nombres de las filas y columnas, hacemos:

```
> k["Dos","Cuatro"]
[1] 5
```

Extrayendo la tercera columna.

```
> k[,3]
   Uno    Dos   Tres Cuatro  Cinco
    -3      7      6      0      0
```

Otra forma de construir matrices es haciendo uso de las funciones `rbind()` y `cbind()`, dando filas o las columnas individuales.

```
> f=rbind(c(1.1,3.5,1.9),c(4.5,3.9,1.2))
> f
     [,1] [,2] [,3]
[1,]  1.1  3.5  1.9
[2,]  4.5  3.9  1.2

> f=cbind(c(1.1,3.5,1.9),c(4.5,3.9,1.2))
> f
```

```
     [,1] [,2]
[1,]  1.1  4.5
[2,]  3.5  3.9
[3,]  1.9  1.2
```

Definamos ahora otra matriz, de la misma dimensión de la matriz `a`.

```
> b=matrix(c(3,5,6,4,0,9),2,3)
> b
     [,1] [,2] [,3]
[1,]    3    6    0
[2,]    5    4    9
```

Para sumar y restar las matrices, hacemos:

```
> a+b
     [,1] [,2] [,3]
[1,]    5    9    4
[2,]    6    6   10

> a-b
     [,1] [,2] [,3]
[1,]   -1   -3    4
[2,]   -4   -2   -8

> b-a
     [,1] [,2] [,3]
[1,]    1    3   -4
[2,]    4    2    8
```

Ahora definamos la matriz `d` como sigue:

```
> d=matrix(c(4,0,2,1,-1,7,4,3,5,3,1,2),3,4)
> d
     [,1] [,2] [,3] [,4]
[1,]    4    1    4    3
[2,]    0   -1    3    1
[3,]    2    7    5    2
```

El producto matricial, se efectúa mediante el siguiente comando.

```
> a%*%d
     [,1] [,2] [,3] [,4]
[1,]   16   27   37   17
[2,]    6    6   15    7
```

La inversa se calcula, mediante:

```
> solve(j)
     [,1]        [,2]
[1,] -0.3  0.26666667
[2,]  0.2 -0.06666667
```

La transpuesta la logramos obtener, mediante:

```
> t(j)
     [,1] [,2]
[1,]    2    6
[2,]    8    9
```

El determinante, lo obtenemos con:

```
> det(j)
[1] -30
```

Los elementos de la diagonal, los podemos obtener de la siguiente manera:

```
> diag(j)
[1] 2 9
```

### 1.5.7. Listas

Una lista en R, es una entidad que consiste en una colección ordenada de objetos, los cuales se conocen como componentes. Las componentes no necesariamente deben ser del mismo tipo.

Las listas pueden componerse de un vector numérico, un valor lógico, una matriz y una función. Para crear una lista, a partir de objetos ya existentes, hacemos uso de la función `list()` y la asignación la hacemos de la siguiente forma:

```
l=list(nombre_1=objeto_1,nombre_2=objeto_2,...,nombre_n=objeto_n)
```

Almencena en "l" una lista de n componentes que son:

```
objeto_1,objeto_2,...,objeto_n
```

A los cuales se les ha asignado los nombres:

```
nombre_1,nombre_2,...,nombre_n
```

Los nombres de los componentes pueden asignarse libremente. Veamos a continuación un ejemplo de lista.

```
> lista=list(nombre="Julissa",no.hermanos=4,
edad.hermanos=c(40,39,38,30))
```

En este caso nuestra lista está compuesta por tres componentes, para referirnos a un componente en particular tecleamos `lista[[]]`, veamos a continuación.

```
> lista[[1]]
[1] "Julissa"
> lista[[2]]
[1] 4
> lista[[3]]
[1] 40 39 38 30
```

La función `length()` devuelve la cantidad de componentes de una lista, si aplicamos esta función en nuestro caso esperamos que devuelva 3, así.

```
> length(lista)
[1] 3
```

Para referirnos a los nombres de los componentes de una lista, lo hacemos mediante la expresión `Nombre de la lista$Nombre del componente`, en nuestro caso:

```
> lista$nombre
[1] "Julissa"
> lista$no.hermanos
[1] 4
> lista$edad.hermanos
[1] 40 39 38 30
```

Vea que `lista$nombre` coincide con `lista[[1]]`, `lista$no.hermanos` con `lista[[2]]` y `lista$edad.hermanos` con `lista[[3]]`.

### 1.5.8. Hojas de datos

Las hojas de datos son listas formadas por vectores de la misma longitud, pero no necesariamente del mismo tipo: unos pueden ser numéricos y otros de tipo factor. Son, de hecho, el contenedor natural de las muestras de datos multivariantes, donde cada vector de la hoja, es una de las variables recogidas, y cada componente de dicho vector es el dato de cada individuo u observación. Así, las hojas de datos se pueden visualizar como una matriz, donde cada columna tiene los datos de una variable en todos los individuos, y cada fila tiene los datos de un individuo en todas las variables.

Las hojas de datos se crean mediante la función `data.frame()`. Veamos a continuación un ejemplo.

Supongamos que tenemos la edad, etnia y califiación de un curso de cinco estudiantes, para crear una hoja de datos hacemos:

```
> ed=c(19,17,18,20,19)
> et=c("Mestizo","Mayagna","Miskito","Mestizo","Mayagna")
> ca=c(56,90,69,95,60)
> curso=data.frame(edad=ed, etnia=et, calif=ca)
> curso
  edad   etnia calif
1   19 Mestizo    56
2   17 Mayagna    90
3   18 Miskito    69
4   20 Mestizo    95
5   19 Mayagna    60
> nombres=c("Arnulfo","Michael","Garel","Luis","Desler")
> row.names(curso) = nombres
> curso
        edad   etnia calif
Arnulfo   19 Mestizo    56
Michael   17 Mayagna    90
Garel     18 Miskito    69
Luis      20 Mestizo    95
Desler    19 Mayagna    60
```

Se observa cómo la función `row.names()` sirve tanto para obtener el vector de nombres de los individuos (filas) de la hoja de datos, como para establecerlos con una asignación.

R posee varias funciones como `read.table()` para leer y archivos de texto en diversos formatos y obtener una hoja de datos con los datos de dichos archivos. Una de las variantes más útiles de la función `read.table()` es la función

`read.csv()`, ya que permite realizar esta operación a partir de archivos que contienen valores separados por comas, uno de los principales formatos de intercambio de información entre manejadores de hojas de cálculo, como Excel.

### 1.5.9. Control de flujo

R, al igual que cualquier otro lenguaje de programación, permite definir acciones dependientes del cumplimiento de determinada condición, o acciones repetitivas que se ejecutarán hasta que se cumpla cierto criterio. Las funciones que permiten el control de flujo son básicamente `for()`, `if()`, `repeat` y `while()` [3].

#### 1.5.9.1. El ciclo `for()`

Se usa para especificar que una determinada operación se repite cierto número de veces, su sintaxis es:

```
for( contador in vector ) {
    ...
}
```

donde `contador` es un nombre, `in` es palabra reservada, y `vector` es un vector ya definido en el programa, de modo que el bloque de código a continuación se va a repetir una serie de veces: la primera vez con la asignación `contador=vector[1]`, la segunda vez con la asignación `contador=vector[2]`, y así sucesivamente hasta terminar de recorrer todo el vector `vector`.

#### 1.5.9.2. El comando `if()`

Lo usamos cuando queremos controlar las operaciones que se deben ejecutar, esto es las que cumplen con cierta condición especificada, su sintaxis es:

```
if( condición ) {
    # orden si cierta
} else {
    # orden si falsa
}
```

donde `condición` es una constante lógica (normalmente el resultado de una comparación entre variables). La parte `else` es opcional.

#### 1.5.9.3. El comando `repeat`

Lo usamos cuando queremos que una determinada sucesión de comandos se repitan, hasta que ejecutemos la orden `break`, su sintaxis es:

```
repeat {
    ...
}
```

Comúnmente incluye la orden `break`; pero se puede hacer uso en `for()` y `while()`.

#### 1.5.9.4. El ciclo `while()`

Se utiliza cuando queremos repetir funciones determinadas; pero en este caso no se fija el número de veces que queremos que las funciones se repitan, su sintaxis es:

```
while( condición ) {
    ...
}
```

donde `condición` es una constante lógica (normalmente el resultado de una comparación entre variables). Si es falsa, el proceso termina. Si la evaluación es verdadera, la orden es ejecutada, el proceso se repite y la expresión es evaluada nuevamente.

### 1.5.10. Funciones

Una función se define asignando a un objeto la palabra `function` seguida de los argumentos que desee dar a la función, escritos entre paréntesis y separados por coma, seguida de la orden, entre llaves si son varias órdenes, que desee asignar a la misma. Entre los argumentos, se destaca que si se utiliza tres puntos seguidos, `...`, ello indica que el número de argumentos es arbitrario [6]. Generalmente las funciones se representan de la siguiente forma:

`nombre = function(arg1, arg2, ...)` expresión

Si queremos saber cuáles son los argumentos de una función dada y sus valores por defecto, usamos `args()`. Por ejemplo la función interna de R que calcula la media aritmética es `mean`, veamos sus argumentos.

```
> args(mean)
function (x, ...)
NULL
```

Para obtener ayuda sobre una función específica (en este caso para la media), tecleamos directamente sobre la consola de R, los siguientes códigos:

```
help("mean")
```

Si queremos más información usamos

```
??mean
```

Los argumentos de una función pueden tomar valores por defecto, a continuación se muestra un ejemplo de función, a la cual hemos nombrado suma ya que calcula la suma de dos objetos que prodrán ser vectores, matrices, variables indexadas, etc.

```
> suma=function(x=20,y=40)
+ {
+ x+y
+ }
> suma()
[1] 60
```

En R existen internamente una serie de funciones, en la Tabla 1.1 se muestran algunas funciones estadísticas y su descripción.

## 1.5.11. Funciones estadísticas

A modo de ejemplo práctico, y de cara al objetivo de proporcionar un menú de R Commander para el cálculo de probabilidades en modelos de variable aleatoria, en la Tabla 1.2 mostramos unas definiciones alternativas para las funciones de (densidad) de probabilidad de algunos de estos modelos. La librería `stats` de R tiene programadas todas estas funciones y muchas más, de modo que no son estrictamente necesarias.

### 1.5.11.1. Intervalo de confianza

La siguiente función calcula el intervalo de confianza para la media poblacional, fue escrita por Larry A. Pace y he personalizado ciertas cosas. Para más detalles, vea [10].

| **Función** | **Descripción** |
|---|---|
| table() | Tabla de frecuencia |
| mean() | Media aritmética |
| median() | Mediana |
| summary() | Estadísticos descriptivos |
| range() | Rango |
| sd() | Desviación estándar |
| var() | Varianza |
| IQR() | Rango intercuartil |
| quantile() | Cuantiles del orden deseado |
| cov() | Covarianza |
| cor() | Coeficiente de correlación |
| lm() | Modelo de regresión lineal |
| glm() | Modelo de regresión lineal generalizado |
| t.test() | Contraste $t$ para una muestra |
| shapiro.test() | Contraste Shapiro-Wilk para una muestra |
| z.test() | Contraste $z$ para una muestra |
| t.test(muestra,mu,conf.level) | Contraste (bilateral) de hipótesis |
| anova() | Tabla del análisis de varianza |

Tabla 1.1: Funciones estadísticas

```
Confint = function(x,alpha=0.05) {
    #x vector de datos, alpha=0.05 (por defecto)
    conflevel=(1-alpha)*100 #Nivel de confianza
    stderr<-sd(x)/sqrt(length(x)) #Error estándar de la media
    tcrit<-qt(1-alpha/2,length(x)-1) #Valor crítico de t
    margin<-stderr*tcrit #Margen de error
    lower<-mean(x)-margin #Cálculo del límite inferior
    upper<-mean(x)+margin #Cálculo del límite superior
    cat("Intervalo de confianza",conflevel,"%","\n")
    cat("Media:",mean(x),"\n")
    cat("Error estándar de la media:",stderr,"\n")
    cat("Límite inferior:",lower,"\n")
    cat("Límite superior:",upper,"\n")
    }
```

```
binomial = function(x,n,p) {
    # x: valor del que se desea la probabilidad
    # n: repeticiones del experimento
    # p: probabilidad de un éxito
    return( choose(n,x)*(p^x)*((1-p)^(n-x)) )
}

poisson = function(x,mu) {
    # x: número de veces que ocurre el evento
    # mu: promedio de ocurrencias por unidad de tiempo o espacio
    return( exp(-mu)*mu^x/factorial(x) )
}

exponencial = function(t,mu) {
    # t: lapso de tiempo
    # mu: intensidad del proceso
    return( 1-exp(-mu*t) )
}

normal = function(x, mu, sigma) {
    # x: valor del que se desea la densidad de prob.
    # mu: media
    # sigma: desviación típica
    return( 1/sqrt(2*pi) * exp( -(x-mu)^2 / 2) )
}
```

Tabla 1.2: Definiciones alternativas de algunas funciones de (densidad) de probabilidad, a modo de ejemplo.

# Capítulo 2

# El paquete R Commander

## 2.1. Aspectos básicos de R Commander

Commander (abreviado Rcmdr) es un paquete de R que proporciona una interfaz gráfica con licencia pública general y fue creado y aún mantenido por John Fox en el departamento de sociología de la McMaster University. Para más información puede visitar la página de R Commander `http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/`.

R Commander fue originalmente desarrollado para para crear gráficos de estadística básica, tiene su base en el paquete tcltk.

El paquete tcltk de R, permite acceder a la plataforma independiente del lenguaje de programación Tcl y a los elementos de la interfaz gráfica de usuario Tk mediante los denominados TkWidgets, ya que el lenguaje tcl/Tk se encuentra embebido en R. Además permite configurar e inicializar el intérprete Tcl a través de funciones específicas de R. Así mediante un método de comunicación se definen pequeñas funciones que toman cadenas de R y pasan a Tcl para su ejecución [2].

Tcl/Tk y el paquete tcltk están disponibles para cualquier sistema operativo bajo los cules funciona R. De esto, la interfaz gráfica de usuario de R Commander corre bajo todas esas plataformas.

La interfaz gráfica de Rcmdr consiste de una serie de botones los cuales nos permiten trabajar directamente sin hacer uso de comandos. Inicialmente no está disponible, por lo que debemos descargarlo e instalarlo, para esto hacemos lo siguiente: iniciamos sesión en R, luego:

**Paquetes → Instalar paquetes(s)....**

Otra manera alternativa de descargar R Commnader es tecleando sobre la consola `install.packages("Rcmdr")`.

Una vez descargado R Commander, iniciamos sesión en R mediante el menú:

**Paquetes → Cargar paquete ... → Rcmdr**

O bien, tecleamos sobre la consola `library(Rcmdr)` y se inicia la interfaz gráfica de R Commander, la cual se muestra en la Figura 2.1.

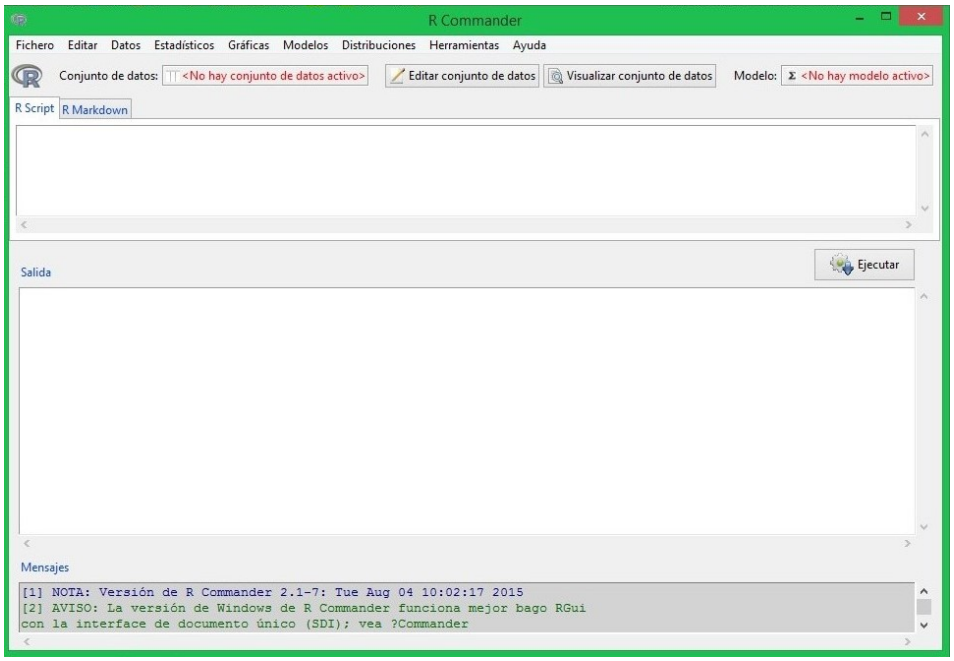


Figura 2.1: Pantalla de Rcmdr

En la Figura 2.1, podemos apreciar que la ventana de R Commander se divide en tres subventanas, las cuales son:

- **R Script:** Aparecen las instrucciones R generadas por la GUI de R Commander, en esta ventana podemos escribir también las instrucciones que queremos realizar, esto lo hacemos a través de comandos de R; pero recordemos que el propósito de R Commander es el evitar tener que escribir códigos.

- **Salida:** Aquí aparecen impresos los resultados, por supuesto, cuando hemos ejecutado comandos en la primera ventana (R Script).

- **Mensajes:** En esta ventana se muestran los mensajes de error, advertencias y el mensaje de inicio (ver [1] NOTA:, de la Figura 2.1)

Cuando creamos gráficos, éstos aparecen en una ventana a parte denominada R Graphics: Device.

El script puede guardarse y volver a ser ejecutado directamente otras veces con otros conjuntos de datos diferentes, sin que el usuario tenga que desplazarse por todo el sistema de menús para volver a realizar las mismas tareas [15].

En la primera línea del menú de la ventana de Rcmdr aparecen las siguientes opciones:

- **Fichero:** Permite «Cambiar» (establecer) el directorio o carpeta de trabajo, «Abrir» un archivo de instrucciones, «Guardar las instrucciones» en diversos formatos, «Guardar los resultados», «Guardar el entorno de trabajo de R» y «Salir» de R Commander [8].
- **Editar:** Opciones de cortar, copiar, pegar, deshacer, limpiar la ventana, etc.
- **Datos:** Presenta utilidades para la gestion de datos (creación de un conjunto de datos, importación desde otros programas, recodificación de variables, tipificar variables, etc.)
- **Estadísticos:** Aquí damos el tratamiento estadístico a los datos a partir de un submenú que contiene resumen numérico de los datos, crear distribuciones de frecuencias, tablas de contingencia, test de proporciones para una y dos muestras, test no paramétricos, modelos de regresión, etc.
- **Gráficas:** Contiene diferentes gráficos: secuencial (para series temporales), histogramas (para variables continuas), de tallo y hojas, diagrama de cajas, diagrama de dispersión (para ver la relación entre dos variables cuantitativas), gráficas de líneas, de barras, de sectores (para variables cualitativas), gráficos 3D, etc. [8].
- **Modelos:** Permite realizar múltiples operaciones relativas a un modelo construido previamente, como elaborar una tabla ANOVA, comparar varios modelos y elegir el más adecuado los criterios de información (AIC, BIC), aplicar modelos paso a paso (por ejemplo el Análisis de Discriminate), construir contrastes, y realizar dianósticos (FIV, Test de Breusch-Pagan para heteroscedasticidad, Test de Durbin-Watson para autocorrelación, diagnósticos de gráficos de residuos). Para que aparezcan todas las opciones del menú es necesario que anteriormente se realizara algún modelo, por ejemplo haber realizado un ANOVA [8].
- **Probabilidades:** Cuantiles, Gráficas y muestra de distribuciones de probabilidad continuas y discretas.

- **Herramientas:** Cargamos paquetes relacionados con Rcmdr, además tenemos un botón de opciones que nos permite cambiar la fuente, tamaño y color de la fuente de la interfaz de Rcmdr y otras opciones.

- **Ayuda:** Enontramos el sitio oficial de Rcmdr e información (en español e inglés) relacionada con éste.

Además de los menús y diálogos, la interfaz de Rcmdr contiene una barra de herramientas (situada debajo de la barra de menús) que contiene los elementos:

- **Conjunto de datos:** Muestra el nombre del conjunto activo de datos. Inicialmente no hay ningún conjunto de datos activos, si presionamos este botón, podremos elegir entre el conjunto de datos que están actualmente en la memoria.

- **Editar conjunto de datos:** Editamos el conjunto de datos activo, agregando o eliminando filas o columnas, además podemos editar los datos que hemos ingresado en la primera ocasión.

- **Visualizar conjunto de datos:** Nos permite ver el conjunto de datos activo, si presionamos este botón, vemos las variables y los datos que se han recolectado.

- **Modelo:** Muestra el modelo estadístico activo, aunque tengamos un conjunto de datos activo, este botón está inactivo, si hay más de un modelo en la memoria podemos elegir entre ellos pulsando sobre el botón.

Una vez cargado el paquete Rcmdr, la ventana de éste inicia al frente de la consola de R, la cual podemos minimizar, podemos ajustar o maximizar la pantalla de R Commander, esto nos permitirá trabajar de manera más cómoda, además los resultados serán más fáciles de visualizar. Podemos realizar cambios de la configuración inicial de Rcmdr, a partir del menú:

**Herramientas → Opciones ...**

Las limitaciones de R Commander consisten en que muchas técnicas estadísticas no están en el menú (solo las más habituales), y que algunas opciones de las técnicas incluidas no se pueden aplicar desde el menú. El conocimiento del lenguaje de programación de R permite multiplicar la potencia estadística y mejorar los resultados y su presentación, para lo cual se pueden modificar convenientemente las órdenes generadas inicialmente por R Commander [7].

## 2.2. Introducir datos

A partir del menú Datos, existen varias formas de introduccir datos en R Commander, veamos a continuación:

- **Nuevo conjunto de datos:** Introducimos datos directamente en el editor de datos, el cual es similar a una hoja de cálculo de excel; pero funciona para un conjunto de datos no muy grande.
- **Cargar conjunto de datos:** Nos permite cargar un conjunto de datos que tengamos en nuestro ordenador, puede ser un archivo de datos de R, SPSS y otros formatos.
- **Importar datos:** Nos permite introducir datos a Rcmdr desde: archivo de texto, portapapeles o URL, datos SPSS, un archivo SAS exportado, datos Minitab, datos STATA y un archivo de excel. Para leer archivo de texto, portapaples o URL, Rcmdr usa por defecto el espacio en blanco como separador de campo y el punto para carácter decimal. Lo podemos cambiar si lo deseamos.
- **Conjunto de datos en paquetes:** Primero nos da opción para cargar un conjunto datos a partir de una lista en paquetes que tiene internamente Rcmdr, allí podemos escoger el que queramos, la otra opción es leer un conjunto de datos desde paquete adjunto.

## 2.3. Resúmenes numéricos

Cuando ya tenemos activo un conjunto de datos, podemos usar los menús de Rcmdr para generar resúmenes numéricos y gráficas. Si queremos un resumen del conjunto de datos activo seleccionamos:

**Estadísticos → Resúmenes → Conjunto de datos activo...**

Con esta opción obtenemos información de la observación más pequeña y más grande, el primer y tercer cuartil, la media y la mediana.

Podemos también crear un resumen numérico de los datos para esto usamos:

**Estadísticos → Resúmenes → Resúmenes numéricos...**

A continuación se abre un cuadro de diálogo que muestra únicamente las variables numéricas.

Seleccionamos la variable de nuestro interés [1] y obtenemos información sobre la media, la desviación, el rango intercuartílico y los cuartiles del conjunto de datos.

Para variables cualitativas podemos crear distribuciones de frecuencias. Para esto usamos:

**Estadísticos → Resúmenes → Distribuciones de frecuencia...**

Seleccionamos la variable de interés y nos aparece el conteo (frecuencia) y el porcentaje de cada clase.

## 2.4. Crear Gráficas

R Commander dispone de varios tipos de gráficas como: gráfica secuencial, histograma, gráfico de tallos y hojas, diagrama de caja, gráfica de comparación de cuantiles, diagrama de dispersión, diagrama de puntos y gráficos 3D.

Para realizar gráficos hacemos uso el menú **Gráficas**, por ejemplo para construir un histograma usamos:

**Gráficas → Histograma...**

Seleccionamos la variable, el histograma a parece a la derecha de la consola de R en la ventana R Graphics Device. Si en una determinada sesión realizamos varias gráficas, éstas se van solapando en la ventana R Graphics Device, quedando al final la más reciente.

R Commander dispone de dos formatos para guardar las gráficas. Si queremos guardar alguna gráfica, hacemos:

**Gráficas → Guardar gráfico en archivo...**

Seleccionamos uno de los dos formatos: como mapa de bits o como PDF/ Postscript/EPS. Si elegimos el primer formato debemos seleccionar entre PNG y JPEG para el formato del gráfico. El segundo formato nos guarda directamente la gráfica en PDF.

[1] Para seleccionar una sola variable en la lista de variables del cuadro, sólo pulsamos sobre su nombre. Para seleccionar más de una variable mantenemos presionada la tecla control y pulsamos sobre el nombre de las variables.

## 2.5. Modelos estadísticos

En R Commander podemos ajustar varios modelos estadísticos, mediante las opciones de menú:

**Estadísticos → Ajuste de modelos**

Entre los modelos tenemos: regresión lineal, modelo lineal, modelo lineal generalizado, modelo logit multinomial y modelo ordinal de regresión.

Si elegimos un modelo lineal o un lineal generalizado se despliega un cuadro de diálogo en el que nos pide introducir la fórmula del modelo.

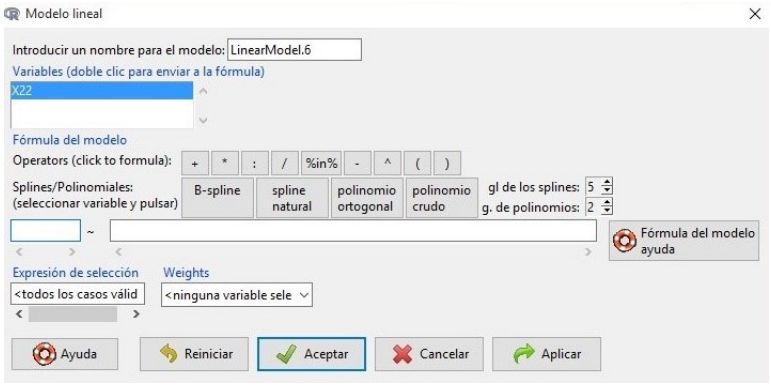


Figura 2.2: Cuadro de diálogo del modelo lineal

Podemos poner nombre al modelo aunque Rcmdr asigna uno por defecto en nuestro caso **Linearmodel.6**. Debemos seleccionar las variables y las agregamos a la fórmula dando doble clic sobre ellas, éstas se agregan al lado izquierdo de la fórmula, si está vacío, de otra forma las agrega al lado derecho.

Las variables categóricas aparecen etiquetadas como tal entre corchetes en la lista de variables.

Sobre la fórmula hay disponible una fila de botones la cual usamos para introducir operadores y paréntesis en el lado derecho de la fórmula.

El cuadro **Expresión de selección** podemos escribir una expresión R, la cual pasa al argumento subset de la función `lm` y se usa para ajustar el modelo a un subconjunto de las observaciones en el conjunto de datos.

Si aún no tenemos fórmula para nuestro modelo, podemos dar clic en el botón Fórmula del modelo ayuda y nos manda a una página web[2] donde se muestra: descripción, uso, argumentos, detalles y ejemplos de los modelos.

[2]La página funciona correctamente sin conección a internet.

## 2.6. Salir de Rcmdr

Existen tres maneras de salir de R Comander:

- Cerrando directamete la ventana, dando clic en el ícono "**X**" que se encuentra en la parte superior derecha.
- Desde el menú **Fichero** → **Salir** → **De Commander**, nos pregunta: si deseamos salir, guardar el archivo de instrucciones, guardar el archivo R Markdown (no entramos en su utilidad en este trabajo) y guardar el archivo de salida. De esta manera terminamos nuestra sesión en Rcmdr; pero seguimos en R.
- Desde el menú **Fichero** → **Salir** → **De Commander y R**, nos pregunta: si deseamos salir, guardar el archivo de instrucciones, guardar el archivo R Markdown y guardar el archivo de salida. De esta manera terminamos nuestra sesión simultáneamente en Rcmdr y R.

## 2.7. Paquetes Plug - In

En la ayuda obtenida mediante `help(Plugins)` encontramos que: "un plug-in de R Commander es un paquete común que (1) proporciona extensiones de los menús de R Commander en un achivo llamado `menus.txt` localizado en el directorio `etc` del paquete; (2) proporciona funciones call - back requeridas para esos menús; y (3) en un campo llamado Model, en el archivo DESCRIPTION del paquete, incrementan la lista de modelos reconocidos por R Commander. Los menús proporcionados por un paquete plug-in se unen con los menús estándar Commander".

A partir de la versión 1.3-0, Rcmdr dispone de alternativas para extensión por paquetes "plug-in" [3], los cuales incrementan el menú de Rcmdr y pueden proveer cajas (adicionales) de diálogos y funcionalidades estadísticas.

Cuando los paquetes plug-in se instalan en el sistema, son detectados automaticamente al arrancar R Commander y pueden ser leídos desde el menú de herramientas de R Commander.

Alternativamente, los plug-in son leídos independiente por las librerías de comandos de R, en este caso, el paquete Rcmdr podrá ser leído con los plug-in's del menú instalados en la barra del menú de R Commander.

Finalmente, los plug-in son leídos automaticamente junto con el paquete Rcmdr configurando las opciones de los plugins del Rcmdr, por ejemplo el comando

[3]Paquetes estándar de R que se desarrollan, mantienen, distribuyen e instalan independientemente del paquete Rcmdr.

```
options(Rcmdr=list(plugins="RcmdrPlugin.TeachingDemos"))
```

hace que el paquete plug-in **RcmdrPlugin.TeachingDemos** sea leído cuando R Commander arranca [5].

Hay disponible una serie de plugins que proporcionan acceso directo a los paquetes de R, a través de la interfaz de Rcmdr. éstos son descargados e instalados de la misma forma que cualquier otro paquete de R, en el CRAN mirror existen 37 plugins, una característica principal para identificarlos es que todos comienzan con el nombre RcmdrPlugin.# seguido del resumen del nombre.

Los plugins se pueden cargar desde la consola de R o usando los menús de Rcmdr:

**Herramientas → Cargar Plugin(s) de Rcmdr**.

En particular RcmdrPlugin.BCA es un plugin para Bussines and Customer Analytics. Para descargarlo podemos hacer como con cualquier paquete: mediante menú o línea de comandos (ver página 4).

Iniciamos sesión en Rcmdr y lo cargamos desde el menú herramientas, una vez seleccionado nos solicita permiso para reinicio con el plugin incorporado.

Una vez confirmado, vemos la nuevainterfaz de Rcmdr en la Figura 2.3[4].

Note que el plugin RcmdrPlugin.BCA agrega los menús necesarios para trabajar la temática para lo que fue creado, esto no es coincidencia ya que cada uno de los restantes plugins trabajan de esta manera.

[4]Vea la diferencia de los menús entre la Figura 2.1 y la Figura 2.3

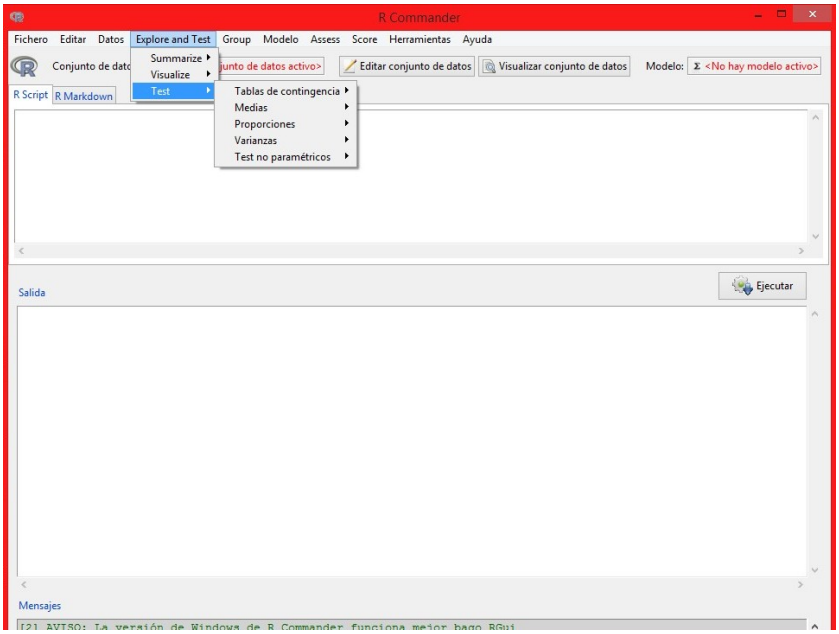


Figura 2.3: Menús de RcmdrPlugin.BCA

# Capítulo 3

# Extensión de R Commander

Un usuario en particular puede modificar el código fuente[1] del paquete R Commander y reconstruirlo. Sin embargo, podemos alterar los menús existentes e incluso agregar nuevos menús al paquete R Commander, sin reconstruirlo. Veamos a continuación dos maneras de hacerlo:

- Modificando el archivo `Rcmdr-menus.txt`[2]: Es un archivo de texto plano que contiene los menús estándar de R Commander, reside en el subdirectorio `etc` del paquete Rcmdr.
- Creando archivos con extensión `.R`: Se originan en el directorio `etc` y son leídos internamente cuando R Commander inicia. En consecuencia, las variables y funciones definidas en estos tipos de archivos (`.R`) están disponibles en el entorno global.

## 3.1. Definición del archivo menú

En Windows, los paquetes que instala un usuario se ubican, de forma automática, en la carpeta del usuario que lo instala. Si el sistema operativo se halla en la unidad "`C:`", el nombre de usuario es "`Osmar`" y se tiene la distribución de R en su versión "`3.2.2`", entonces los nuevos paquetes instalados se sitúan en:

```
C:\users\Osmar\Documentos\R\win-library\3.2\
```

Con una subcarpeta con el nombre de cada paquete instalado, como "`Rcmdr`".

[1] Es un archivo de texto ASCII que contiene líneas de código de un programa. El compilador toma este archivo, lo analiza y produce un archivo en lenguaje de máquina, que el ordenador puede ejecutar.

[2] `Rcmdr-menus.txt` es el archivo fuente de los menús de R Commander.

El archivo que gestiona los menús de R Commander es `Rcmdr-menus.txt` y se sitúa en:

```
C:\users\Osmar\Documentos\R\win-library\3.2\Rcmdr\etc
```

El archivo contiene siete entradas (campos) y definen ya sea un menú o un ítem de menú, cada uno de los campos están separados por espacios en blancos (vea la Figura 3.1).

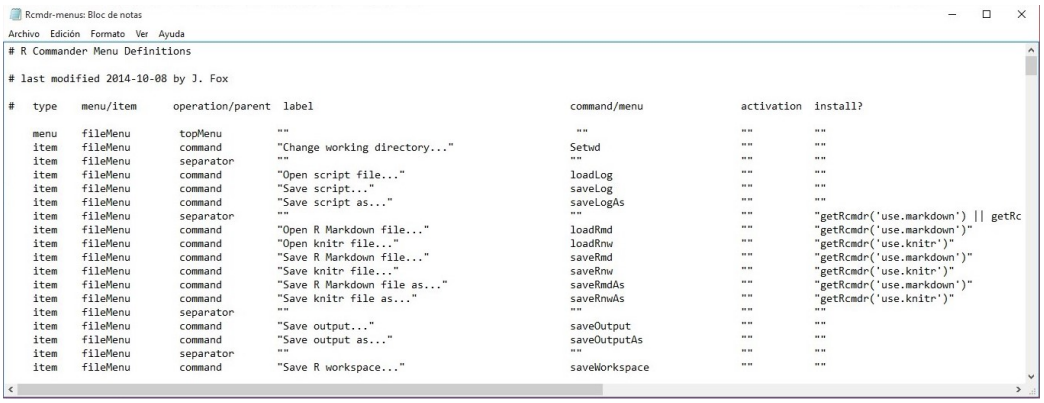


Figura 3.1: Archivo Rcmdr-menus

A continuación daremos una breve descripción de cada una de las entradas.

1. **type:** Se especifica el tipo de entrada que vamos a añadir, pueden ser:

   - menu: Se utiliza siempre que vamos a insertar un menú principal, también puede ser un submenú. No hay diferencia entre menú principal y submenú, los dos son denotados con menu; cabe mencionar que su construcción en las columnas no es la misma.
   - item: Cuando la entrada ejecuta una aplicación directamente.

2. **menu/item:** Aquí hay tres opciones de entrada

   - Empezamos con un menú o submenú, en cuyo caso escribiremos en la columna un nombre interno que distinga cada menú. Comúnmente son nombres del tipo: filemenu, datamenu,etc.
   - Agregamos un nuevo item al menú. En esta columna se pondrá el nombre del menú o submenú al que pertenece el item.

- La última opción es cerrar un menú o submenú.

3. **operation/parent:** Tenemos tres posibilidades de entrada
   - Cuando iniciamos un menú o submenú indicaremos su localización. Si es un submenú pondremos el nombre interno del menú que lo contiene, si es un menú principal escribimos la orden topmenu.
   - Si es un item del menú que ejecuta una orden escribimos la instrucción command.
   - Para cerrar un menú o submenú escribimos la orden cascade.
4. **label**:
   - Cuando iniciamos un menú o submenú, se pondrán comillas vacías " ".
   - Si es un item del menú escribimos la etiqueta (dentro de comillas) que queremos que aparezca en el programa.
   - Para cerrar un menú o submenú escribimos (dentro de comillas) en esta columna la etiqueta del menú o submenú.
5. **command/menu:**
   - Cuando iniciamos un menú o submenú, se pondrán comillas vacías " ".
   - Si es un item del menú que ejecuta una orden escribimos la instrucción que queremos llamar.
   - Para cerrar un menú o submenú escribimos el nombre del menú interno que queremos cerrar.
6. **activation:** Contiene una expresión de R que, cuando es evaluada indica que el menú o item del menú está activo, si la expresión es `TRUE`, o inactivo (en gris) si es `FALSE`. Si escribimos comillas vacías se entenderá que esa entrada estará siempre activa.
7. **install?:** Se especifica si ha de cargar un paquete concreto para que funcione la aplicación a la que llama el item. "`packageAvailable(nombre del paquete)`" es la orden que se debe usar. En caso de ser un menú, este campo se rellena con comillas vacías.

Para analizar el estado actual de R Commander, el paquete Rcmdr exporta un determinado número de funciones, entre las que hallamos para: configurar y

recuperar información, determinar la activación de un menú o ítem, construir los elementos de un cuadro de diálogo, funciones varias y funciones estadísticas.

El estado de los menús es evaluado al iniciar R Commander, es reevaluado cuando el conjunto de datos o los modelos estadísticos activos cambian, y siempre que la función `activateMenus` es invocada.

## 3.2. Construyendo un nuevo menú

Construiremos un pequeño menú haciendo uso de los aspectos teóricos descritos anteriormente y luego lo agregamos al menú de R Commander.

Para comenzar, lo primero es tener una idea clara de la estructura que debe contener el menú que deseamos. A manera de ejemplo, se propone el menú con la estructuctura mostrada en la Figura 3.2.

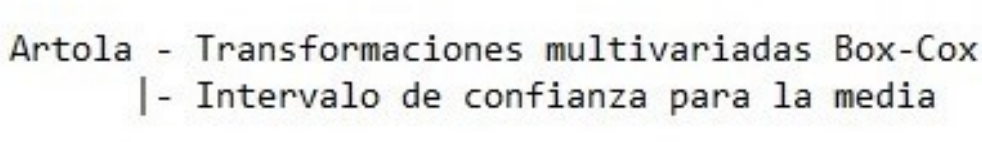


Figura 3.2: Menú Artola

Ahora debemos ubicarlo dentro del menú de Rcmdr, en mi caso particular deseo verlo a la izquierda de **Fichero**, así que debemos editar el inicio del fichero `Rcmdr-menus.txt`, en la Figura 3.3 mostrada abajo ya se ha insertado el nuevo menú.

Observemos que en el campo `command/menu` se ha puesto el nombre de la aplicación que inicia la ventana, la cual se llama `BoxCox`.

Una vez guardados los cambios en el archivo `Rcmdr-menus.txt`, iniciamos sesión en R Commander y si todo ha ido bien, veremos el nuevo menú que hemos agregado, vea la Figura 3.4.

## 3.3. Añadir una función a R Commander

Mostraremos cómo añadir una función al menú de R Commander. Para lograrlo debemos primero crear la función, en otras palabras programarla en lenguaje R, la extensión del archivo con la cual debemos guardar nuestra aplicación debe ser `.R`, un ejemplo de nombre puede ser `artola.R`.

Debemos construir una ventana de diálogo, con sus etiquetas, botones y entradas en un archivo de texto en lenguaje `tcl-tk` y guardarlo con extensión `.R`.

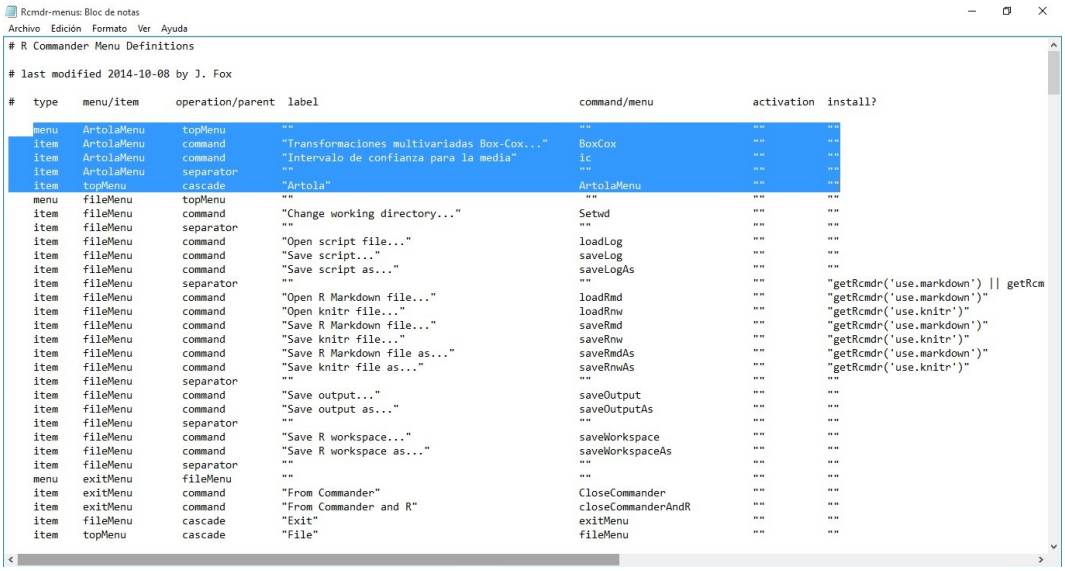


Figura 3.3: Editando el archivo Rcmdr-menus.txt

A continuación se muestra el código que nos permite crear la ventana en R Commander de la función transformaciones mutivariadas Box-Cox, para más información vea [4].

```
BoxCox = function() {
  initializeDialog(title="Transformaciones Box-Cox")
  variablesBox<-variableListBox(top,Numeric(),
    selectmode="multiple",
    title="Seleccionar variables(una o más)")
  onOK<-function(){
    variables<-getSelection(variablesBox)
    if(length(variables)<1){
      errorCondition(recall=BoxCox,
      message="Debe seleccionar una o más variables.")
      return()
    }
    closeDialog()
    command<-paste("box.cox.powers(na.omit(cbind(",
      paste(paste(variables,"=",ActiveDataSet(),"$",
      variables,sep=""),
      collapse = ","),")))",sep="")
    doItAndPrint(command)
```

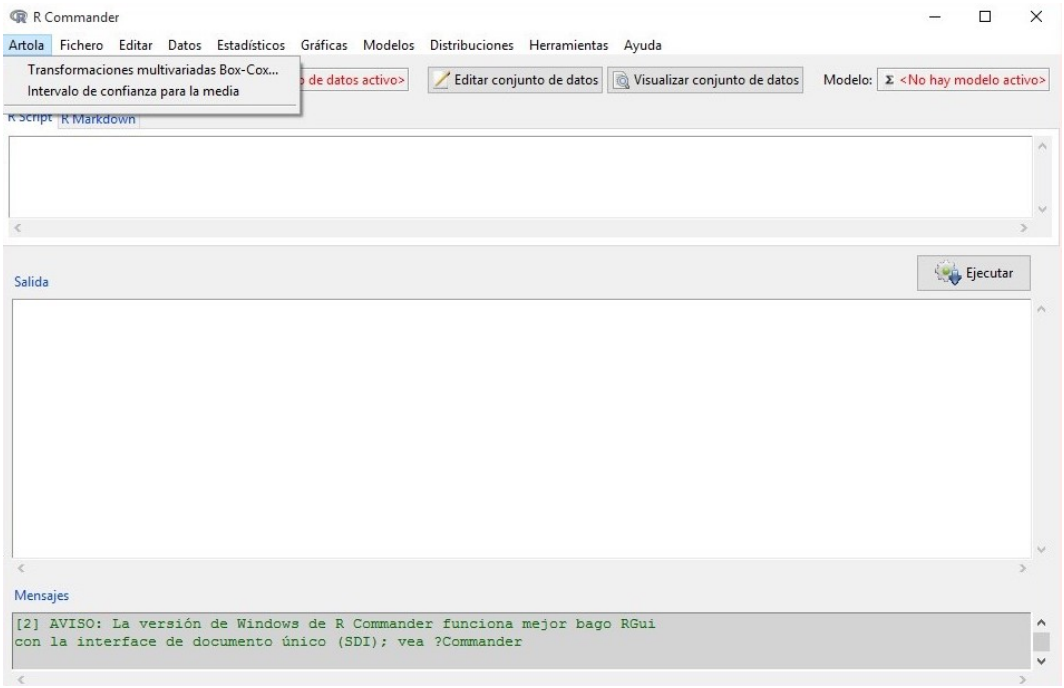


Figura 3.4: Menú agregado a Rcmdr

```
    tkfocus(CommanderWindow())
  }
  OKCancelHelp(helpSubject="box.cox.powers")
  tkgrid(getFrame(variablesBox),sticky="nw")
  tkgrid(buttonsFrame,sticky="w")
  dialogSuffix(rows=2,columns=1)
}
```

Ahora mostramos la ventana de diálogo para la función `Confint` que hemos presentado en 1.5.11 y que está en nuestro menú.

```
ic = function(){
  initializeDialog(title=gettext("Intervalo de confianza"))
  .numeric<-Numeric()
  variablesBox<-variableListBox(top,.numeric,selectmode="multiple",
  title=gettextRcmdr("Seleccionar variables(una o más)"))
  onOK<-function(){
    variables<-getSelection(variablesBox)
    if(length(variables)<1){
```

```
      errorCondition(recall=ic,
        message=gettextRcmdr("Debes seleccionar una o más
          variables."))
      return()
    }
  level=tclvalue(alphaLevel)
  closeDialog()
  doItAndPrint(paste("Confint(na.omit(cbind(",
    paste(paste(variables,"=",ActiveDataSet(),"$",variables,
    sep=""), collapse=","),")))", sep=""))
  tkfocus(CommanderWindow())
  }
  OKCancelHelp(helpSubject="Confint")
  alphaFrame=tkframe(top)
  alphaLevel=tclVar("0.05")
  alphaField=tkentry(alphaFrame,width="6",textvariable=alphaLevel)
  tkgrid(getFrame(variablesBox), sticky="nw")
  tkgrid(tklabel(alphaFrame,
  text=gettextRcmdr("Introduzca el valor de alpha"),fg="blue"))
  tkgrid(alphaField,sticky="w")
  tkgrid(alphaFrame, sticky="w")
  tkgrid(buttonsFrame, columnspan=2,sticky="w")
  dialogSuffix(rows=2,columns=1)
}
```

Debemos tener presente que las funciones que generan una ventana no sólo construyen la ventana, sino que le dicen a R qué función debe ejecutar cuando el usuario pulsa el botón Ok de la ventana.

El paquete Rcmdr imprime en pantalla todas las órdenes que el usuario ejecuta mediante su entorno. De esta forma podemos ver las órdenes que estamos utilizando y con qué argumentos, de esto se encarga la orden `doItAndPrint`.

Para tener resultados exitosos debemos guardar el script `BoxCox.R` en el directorio:

```
C:\users\Osmar\Documentos\R\win-library\3.2\Rcmdr\etc
```

que Rcmdr carga al inicio.

Cuando todo está listo, iniciamos sesión en R Commander y hacemos uso de la funciones, a partir del menú `Artola`, aparece la ventana de diálogo mostrada en la Figura 3.5.

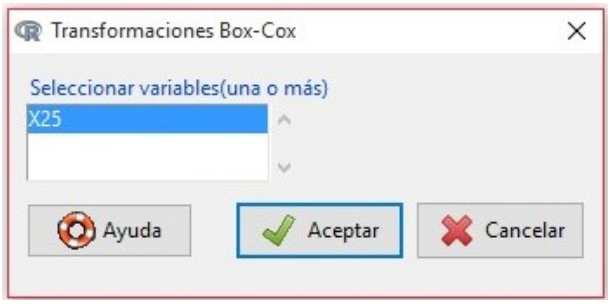


Figura 3.5: Ventana de diálogo para la función Box-Cox

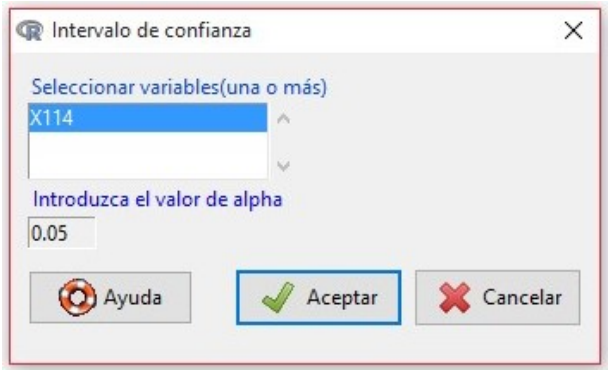


Figura 3.6: Ventana de diálogo para la función Confint

Cabe mencionar que, cuando introducimos datos a R Commander, las funciones `Box-Cox` y `Confint` del menú `Artola` funcionan correctamente.

## 3.4. Creación de paquetes Plug-In para R Commander

Antes de la versión 1.3.0 de R Commander, la única forma de extender los menús programados era la que se ha mostrado en las secciones anteriores. Para ofrecer al público el R Commander con los nuevos menús, era necesario recompilar el código fuente, creando así un nuevo paquete.

Por ello, John Fox, autor de R Commander, redefinió la estructura de su paquete para facilitar la tarea de extensión de menús. Las extensiones serían realizables por medio de paquetes independientes "normales" (de los que aportan nuevas fun-

ciones), y que se pudieran instalar en R sin depender de R Commander. Una vez instalados, R Commander los detectaría, permitiría al usuario "cargar" el paquete, y con el reinicio de R Commander aparecerían los nuevos menús.

Para facilitar la búsqueda de extensiones de R Commander en CRAN, John Fox recomienda llamar a los paquetes plug-in con el nombre `RcmdrPlugin.XXX`, donde `XXX` representa la porción de nombre que su autor desee. De esta forma, al buscar en el listado, aparecerán todos juntos, tras el paquete `Rcmdr`.

### 3.4.1. Creación de un paquete de R en Windows

Para empezar, se debe crear un paquete de R que contenga las funciones que van a ponerse en marcha en cada una de los nuevos botones de menú que se definan.

La creación de paquetes en R es un proceso complejo, véase [11] en la documentación oficial de R. Es una referencia muy completa pero demasiado técnica para los no informáticos. Afortunadamente, existen documentos simplificados para la creación de paquetes sencillos en Windows [9, 13, 12] y, para más ayuda, el programa RStudio [14] simplifica y automatiza los pasos necesarios, cuando se trata de un paquete sencillo.

Así, para ilustrar el proceso, creamos el paquete que por defecto crea RStudio desde su interfaz

- Menú: `New project > New directory > R Package`.
- Elegimos Tipo: `Package`, Nombre: `Artola`.
- Elegimos una carpeta donde crearlo.
- `Create project`.
- Se abre RStudio con un archivo ya creado y completo, llamado "hello.R", en la ventana de edición, que contiene la definición de una función sencilla llamada "hello()".
- Observamos que se ha creado la carpeta "Artola" en el lugar indicado del disco, con la estructura:
  - Carpeta "man" (con el archivo "hello.Rd", de documentación de ayuda). Para cada archivo ".R" que incorpore la definición de una o varias funciones, debe documentarse un archivo ".Rd" que explique los detalles del uso de la función.
  - Carpeta "R" (con el archivo "hello.R", scripts de la función definida). En esta carpeta deben incluirse todos los archivos ".R" que definen nuevas funciones.

  - Archivo DESCRIPTION (informativo del uso del paquete, su autor, etc.). Se debe editar con un editor de texto para informar de la utilidad, el autor,etc.
  - Archivo NAMESPACE. Define el espacio de nombres para que las funciones no entren en conflicto con otras funciones de otros paquetes.
  - Otros archivos propios de RStudio (el del projecto y otro).

- Se construye el nuevo paquete y se carga en la sesión (por medio de Ctrl + Shift + B).
- La función "hello()" está disponible para el usuario.
- Se chequea el paquete (Ctrl + Shift + E). Comprueba que no hay errores en los metadatos del archivo DESCRIPTION y NAMESPACE, que no hay errores de sintaxis o campos vacíos en la documentación de ayuda, que funcionan los ejemplos si se han proporcionado, etc. El resultado es "R CMD check succeeded".
- Se testea el paquete (Ctrl + Shift + T), por las dependencias que puede tener de otros paquetes. El resultado sale ERROR, porque no encuentra una carpeta "testthat" para hacer las pruebas. Pero no indica un error real en el paquete creado por defecto.
- Menú `Build > Build binary package`: crea el archivo "`Artola_0.1.zip`", que R puede instalar desde sus opciones de manú.
- Menú `Build > Build source package`: crea el archivo "`Artola_0.1.tar.gz`" que contiene el código fuente del paquete, es decir, la estructura completa de carpetas y archivos, que otro programador puede usar para desarrollar sus iniciativas a partir del paquete original.

El proceso consiste, entonces, en definir la función adecuada que tome los argumentos que el usuario de R Commander va a introducir en una nueva caja de opciones que proporcionará un nuevo botón de menú. Pasamos entonces a la creación del nuevo botón de menú

### 3.4.2. Creación de nuevas opciones de menú por plug-ins de R Commander

Como aparece en el artículo [5, p.51], se debe crear y editar un archivo llamado `menu.txt` en la carpeta `\inst\etc` de la estructura de carpetas del nuevo paquete. Por el momento, dicha carpeta no existe y se crea.

Creamos un fichero de texto llamado `menus.txt` en dicha carpeta, y lo editamos de tal modo que queda únicamente las cabeceras y el texto resaltado en azul de la Figura 3.3.

La funciones `BoxCox` y `ConfInt` se copian en sendos archivos con extensión ".R" en la carpeta `\R` de la estructura de carpetas.

Finalmente se ejecuta el montaje de la librería con ayuda de RStudio y se instala el nuevo paquete.

En el capítulo siguiente se detalla la creación e instalación de un nuevo botón de menú para facilitar el cálculo de probabilidades a usuarios menos expertos.

# Capítulo 4

# Menú para cálculo de probabilidades

El propósito de este capítulo es crear un menú que nos permita realizar cálculos de probabilidades en modelos de variable aleatoria discreta y continua, y que sea algo más natural que el que lleva R Commander por defecto.

Para contribuir a R Commander con nuevos menús, hay que crear un nuevo paquete (de tipo Plugin) que contendrá (opcionalmente) una o varias funciones nuevas, y un archivo "`menu.txt`" (en la carpeta "`\inst\etc`" del nuevo paquete), con la misma estructura que "`Rcmdr-menus.txt`", y que se engranará automáticamente con el mismo, sin modificarlo, cuando el usuario cargue dicho plugin en la sesión de R Commander.

## 4.1. Distribuciones de probabilidad

En el menú actual `Distribuciones` de R Commander, podemos seleccionar entre las siguientes distribuciones de probabilidad:

- **Distribuciones continuas:** normal, t, Chi-cuadrado, F, exponencial, uniforme, beta, Cauchy, logística, lognormal, gamma, Weibull y Gumbel.
- **Distribuciones discretas:** binomial, Poisson, geométrica, hipergeométrica y binomial negativa.

Para cada una de ellas podemos seleccionar: cálculo de cuantiles, cálculo de probabilidades acumuladas, cálculo de probabilidades, realización de gráficas y una muestra de la distribución.

En la Figura 4.1 se muestra el menú desplegado hasta elegir la distribución Binomial.

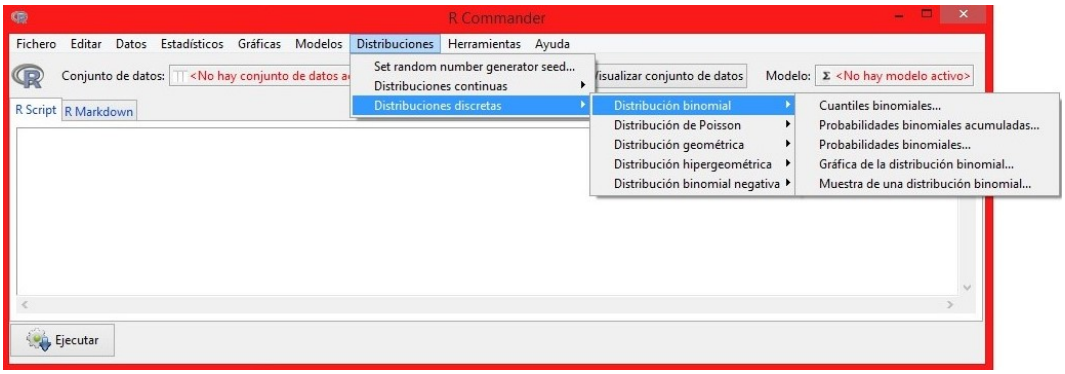


Figura 4.1: Menú de la distribución Binomial

## 4.2. Distribución binomial

Para calcular $P(X > 5)$ con el modelo binomial de parámetros $n = 10$ y $p = 0.3$, lo que hay que hacer con R Commander es elegir:

Distribuciones → distribuciones discretas → distribución binomial → Probabilidades binomiales acumuladas... y aparece la ventana de la Figura 4.2.

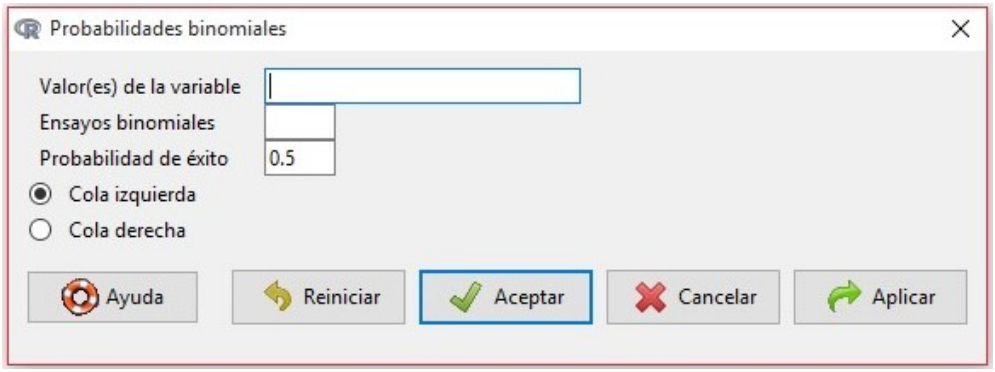


Figura 4.2: Probabilidades binomiales acumuladas

En la ventana de diálogo mostrada arriba, se deben incluir: el valor o valores de los órdenes de cuantiles que queramos calcular (si hay más de uno debe ir separado por comas), el valor de $n$ (número de ensayos) y el valor de $p$, la probabilidad de éxito (por defecto aparece 0.5).

Por último, se debe seleccionar "Cola izquierda", si se quiere calcular la proba-

bilidad de valores a la izquierda del 5 (hasta el 5 inculido). Si se selecciona "Cola derecha", se calculará la probabilidad de los valores a la derecha del 5, sin incluir a éste.

Si queremos calcular $P(X = 8)$, hay que hacer:

Distribuciones → distribuciones discretas → distribución binomial → Probabilidades binomiales... y aparece la ventana de la Figura 4.3.

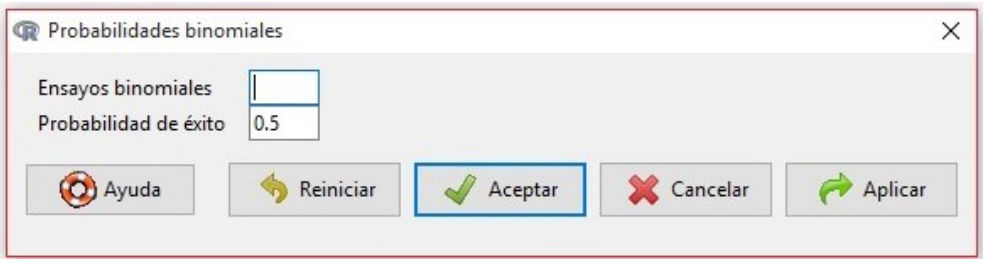


Figura 4.3: Probabilidades binomiales

En la ventana mostrada arriba debemos incluir: el valor de $n$ (número de ensayos) y el valor de $p$, la probabilidad de éxito (por defecto aparece 0.5).

El resultado nos proporciona los valores de la función de masa o probabilidad para todos los valores posibles de la variable.

## 4.3. Nuestro propósito

Las opciones de menú que ofrece la distribución actual de R Commander, para el cálculo de probabilidades de variables aleatorias que siguen un modelo de distribución conocido, son una gran ayuda comparada con la opción de escribir en la consola de R el comando que realiza dicho cálculo.

Sin embargo, pensamos que exigen, del usuario, un conocimiento más que intuitivo de las propiedades de la probabilidad. Entre todas las acciones sobre variables aleatorias, las que corresponden al cálculo de probabilidades, están referidas como "Probabilidades..." y "Probabilidades acumuladas...".

En las situaciones habituales, el usuario reconoce una situación aleatoria concreta, de la que quiere calcular cierto riesgo. El riesgo corresponde a la probabilidad de un suceso, que puede ser de tipo intervalo de valores, o un valor concreto. Por otro lado , la cantidad aleatoria se consigue asimilar a una de las variables aleatorias conocidas e implementadas en R, que corresponde a un modelo y a unos valores concretos de los parámetros de dicho modelo.

Por tanto se busca una probabilidad del tipo $P(X = 0)$ o $P(X \geq 1)$, o $P(10 < X \leq 20)$, entre muchas otras posibilidades sencillas. Formas más complejas implican la combinación (adición) de estos tipos.

Para cierto tipo de usuario (aprendices de una titulación menos técnica o profesionales), la elección del modelo y el planteamiento del suceso cuya probabilidad interesa medir, es un objetivo límite, en el sentido que no dominan matizaciones más allá de dicha modelización. Por eso pensamos que las referencias que hace R Commander, en sus menús, a las colas izquierda y derecha, superan la comprensión de parte de ese público. Y más todavía el matiz de que una cola derecha no incluye el valor que marca dicha cola, lo cual no es despreciable cuando se trata de una variable discreta.

Por ello, para ayudar al colectivo referido, pensamos en crear una interfaz más amigable para el cálculo de probabilidades, de modo que el menú muestre, una pestaña de "probabilidad", donde se ofrecen las opciones:

- $P(X = \square)$
- $P(X \neq \square)$
- $P(X \leq \square)$
- $P(X < \square)$
- $P(X \geq \square)$
- $P(X > \square)$
- $P(\square < X \leq \square)$
- $P(\square < X < \square)$
- $P(\square \leq X < \square)$
- $P(\square \leq X \leq \square)$

y el usuario debe ingresar un valor numérico en los cuadros correspondientes a la probabilidad que le interesa calcular.

Una vez determinado el suceso cuya probabilidad se pretende calcular, en otra pestaña se ofrece el modelo de distribución de la variable en cuestión:

- Binomial de $\square$ intentos con probabilidad de éxito $\square$.
- Poisson de media $\square$.
- Exponencial de media $\square$ (o bien ligada a un proceso de intensidad $\square$).

- Uniforme en el intervalo $[\square, \square]$.
- Normal de media $\square$ y desviación típica $\square$ (o varianza $\square$)

de modo que el usuario debe ingresar valores para los parámetros correspondientes al modelo que sigue su variable de interés.

## 4.4. El plug-in `RcmdrPlugin.Artola`

Para crear las opciones de menú deseadas, creamos un nuevo paquete de R, con la ayuda del software Rstudio, que contiene una única función, capaza de calcular todo.

El lugar más indicado para esta mueva opción de menú es el botón de "Distribuciones". Por ello, el archivo `menus.txt` que debe hacer en la carpeta `\inst\etc` tiene el código:

```
# Rcmdr menus para el paquete Artola

# last modified 2015-09-22 by J. Artola

# type menu/item operation/parent label
    command/menu activation  install?

    # Distributions
    item    distributionsMenu command
    "Calcula probabilidades (Artola)..."
    probArtola                                  ""          ""
```

En esta ocasión se va a implementar una función para calcular las probabilidades del tipo $P(\square < X \leq \square)$. El resto de casos se realizaría de forma muy similar, pero no se ha podido realizar por limitaciones de tiempo.

El paquete contiene un único archivo "probArtola.R", que incorpora la función que se invoca desde el menú, y que representa la caja con las opciones y calcula el valor solicitado.

Tal y como recomienda John Fox en [5], se puede partir de una función del paquete Rcmdr o cualquiera de sus plugins, para adaptar las condiciones de la caja de opciones, ya que el lenguaje `Tcltk` es muy complejo. Es lo que se ha hecho en este caso, y el resultado se halla en Anexo.

## 4.5. Conclusiones

La creación de nuevas opciones de menú para el paquete R Commander puede acercar el uso del R a un mayor público. Ya existen muchas aportaciones de menús especializados, pero en nuestro caso hemos cubierto una opción muy básica, que es el cálculo de probabilidades en modelos conocidos. Ha quedado por implementar algunas variantes de sucesos, pero se trata de una tarea trivial a partir del paquete que proporciona este trabajo.

En la misma línea, hay otras opciones, como son el cálculo de intervalos de confianza, que están cubiertos por menús del R Commander correspondientes al plugin HH, donde las opciones no son del todo claras, y se podría aportar en ese sentido.

Agregar nuevos menús a R Commander es un proceso sencillo, en la parte de que aparezcan en la barra de menús, ya que consiste en editar un pequeño archivo de texto donde se indique en que submenú se situa, a qué función llama si se trata de una opción que lleve a ejecutar un comando, etc.

La parte de definir la ventana de opciones, en la que el usuario elige variables, o introduce valores de parámetros que pasarán como argumentos de la función a ejecutar, es bastante más laboriosa. A pesar de disponer de un gran número de ejemplos en las que ya existen (gracias a que R es de código abierto), si se quiere haer algo original, es preciso acudir a funciones de las librerías `tcltk` y `tcltk2`, donde están las funciones que diseñas al detalle las ventanas.

El objetivo de este trabajo era doble: de una parte profundizar en una herramienta utilizada durante el Máster, como es el software R. Conocer algo más su lenguaje, su estructura modular, y en particular aprender a contribuir al mismo con la creación de un paquete, aunque se trata de un paquete muy simple. De otra parte, se ha pretendido que la contribución pudiese ser útil a la comunidad, de modo que una vez completadas las opciones, se dará a conocer en el ámbito universitario, para que profesorado y alumnado puedan utilizarlo con la esperanza de mejorar rendimiento en el cálculo de probabilidades con R Commander.

# Bibliografía


[1] W. John Braun and Duncan J. Murdoch. *A first Course In statistical Programming With R.* Cambridge University Press, 2007.

[2] Peter Dalgaard. The R-Tcl/Tk interface. 2001.

[3] Paula Elosua. *Introducción al entorno R.* Universidad del País Vasco, 2011.

[4] John Fox. The R Commander: A basic statistics graphical user interface to R. september 2005.

[5] John Fox. Extending the R Commander by "Plug-In" Packages. December 2007.

[6] José Miguel Contreras García, Elena Molina Portillo, and Pedro Arteaga Cezón. *Introducción a la programación estadística con R para profesores.* 2009.

[7] Cástor Guisande Gónzalez and Antonio Vaamonde Liste. *Gráficos estadísticos y mapas con R.* Díaz de Santos, 2012.

[8] Cástor Guisande Gónzalez, Antonio Vaamonde Liste, and Aldo Barreiro Felpeto. *Tratamiento de datos con R, STATISTICA y SPSS.* Díaz de Santos, 2013.

[9] Friedrich Leisch. Creating R packages. Invited tutorial at "COMPSTAT 2008", Porto, Portugal, August 24–29 2008.

[10] Larry A. Pace. *Beginning R. An introduction to Statistical Programming.* Apress®, 2012.

[11] R Core Team. *Writing R Extensions*, 2015. Version 3.2.2(2015-08-14).

[12] B Ripley. Making an R package, 2008.

[13] P Rossi. Making R packages under windows: A tutorial, 2006.

[14] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2015.

[15] Angelo Santana. *Introducción al uso de R-Commander*.

[16] Julio Sergio Santana and Efraín Mateos Farfán. *El arte de programar en R, un lenguaje para estadística*. Instituto Mexicano de Tecnología del Agua, primera edition, 2014.

[17] W. N. Venables, D. M. Smith, and R Core Team. *An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics*, 2015. Version 3.3.0 Under development (2015-07-04).

# Apéndice A

# Anexo

Archivo `probArtola.R`, que contiene la definición de la función `probArtola`, llamada por el botón de menú creado, y que sirve para facilitar el cálculo de probabilidades en modelos.

```
# necesita estas variables globales de Rcmdr
# si la versión de R es posterior a 2.15.1
if (getRversion() >= '2.15.1') globalVariables(c('top', 'buttonsFrame',
        'slider.env', 'notebook'))

# definición de la función que activa el menú
probArtola = function(){
#  require(tcltk)
  require(tcltk2)
  # valores iniciales (por defecto) de los
  # parámetros que el usuario puede variar
  defaults = list(valorinf="-Inf", valorsup="Inf",
                  bin.n="", bin.p="", poi.m="",
                  exp.m="", exp.r="", uni.a="",
                  uni.b="",nor.m="", nor.s="",
                  nor.v="")
  # fabrica la caja de opciones
  dialog.values = getDialog("prob2", defaults)
  # pone título a la caja
  initializeDialog(title=gettextRcmdr("Probabilidades en modelos"),
                   use.tabs=TRUE)

  # definimos las variables que tomarán
  # inicialmente los valores por defecto
```

```
# pero luego cambiarán si el usuario pone otro valor
# (por ejemplo "valorinf")
# a cada variable va asociada una "entry",
# (por ejemplo "valorinf.en")
# que es las "caja" donde el usuario escribe
# el valor del parámetro que le interesa, y que luego
# acaba en la variable

# límite inferior del intervalo
valorinf = tclVar(dialog.values$valorinf)
valorinf.en <- tkentry(top, width="4", textvariable=valorinf)
# límite superior del intervalo
valorsup = tclVar(dialog.values$valorsup)
valorsup.en <- tkentry(top, width="4", textvariable=valorsup)
# binomial: n
bin.n = tclVar(dialog.values$bin.n)
bin.n.en <- tkentry(top, width="6", textvariable=bin.n)
# binomial: p
bin.p = tclVar(dialog.values$bin.p)
bin.p.en <- tkentry(top, width="6", textvariable=bin.p)
# poisson: media
poi.m = tclVar(dialog.values$poi.m)
poi.m.en <- tkentry(top, width="6", textvariable=poi.m)
# exponencial: media
exp.m = tclVar(dialog.values$exp.m)
exp.m.en <- tkentry(top, width="6", textvariable=exp.m)
# exponencial: intensidad proceso
exp.r = tclVar(dialog.values$exp.r)
exp.r.en <- tkentry(top, width="6", textvariable=exp.r)
# uniforme: mínimo
uni.a = tclVar(dialog.values$uni.a)
uni.a.en <- tkentry(top, width="6", textvariable=uni.a)
# uniforme: máximo
uni.b = tclVar(dialog.values$uni.b)
uni.b.en <- tkentry(top, width="6", textvariable=uni.b)
# normal: media
nor.m = tclVar(dialog.values$nor.m)
nor.m.en <- tkentry(top, width="6", textvariable=nor.m)
# normal: desviación típica
nor.s = tclVar(dialog.values$nor.s)
```

```
nor.s.en <- tkentry(top, width="6", textvariable=nor.s)
# normal: varianza
nor.v = tclVar(dialog.values$nor.v)
nor.v.en <- tkentry(top, width="6", textvariable=nor.v)

# función que define la acción a realizar cuando se
# pulsa el botón "OK"
onOK <- function(){
  # cierra caja
  closeDialog()
  # según los parámetros completados, se va a calcular
  # la probabilidad con uno u otro modelo.
  # artesanalmente, se "fabrica" el comando de R completo,
  # como cadena de texto, a base de "paste()",
  # usando las variables definidas anteriormente
  if( tclvalue(bin.n) != "" &
      tclvalue(bin.p) != "" ) {
    comm = paste("pbinom(q=", tclvalue(valorsup), ", size=",
                 tclvalue(bin.n), ", prob=", tclvalue(bin.p),
                 ") - ",
                 "pbinom(q=", tclvalue(valorinf), ", size=",
                 tclvalue(bin.n), ", prob=", tclvalue(bin.p),
                 ")", sep="")
  } else if( tclvalue(poi.m) != "" ) {
    comm = paste("ppois(q=", tclvalue(valorsup), ", lambda=",
                 tclvalue(poi.m), ") - ",
                 "ppois(q=", tclvalue(valorinf), ", lambda=",
                 tclvalue(poi.m), ")", sep="")
  } else if( tclvalue(exp.m) != "" ) {
    comm = paste("pexp(q=", tclvalue(valorsup), ", rate=1/",
                 tclvalue(exp.m), ") - ",
                 "pexp(q=", tclvalue(valorinf), ", rate=1/",
                 tclvalue(exp.m), ")", sep="")
  } else if( tclvalue(exp.r) != "" ) {
    comm = paste("pexp(q=", tclvalue(valorsup), ", rate=",
                 tclvalue(exp.r), ") - ",
                 "pexp(q=", tclvalue(valorinf), ", rate=",
                 tclvalue(exp.r), ")", sep="")
  } else if( tclvalue(uni.a) != "" &
             tclvalue(uni.b) != "" ) {
```

```
      comm = paste("punif(q=", tclvalue(valorsup), ", min=",
                   tclvalue(uni.a), ", max=", tclvalue(uni.b),
                   ") - ",
                   "punif(q=", tclvalue(valorinf), ", min=",
                   tclvalue(uni.a), ", max=", tclvalue(uni.b),
                   ")", sep="")
    } else if( (tclvalue(nor.m) != "") &
               ((tclvalue(nor.s) != "") | (tclvalue(nor.v) != "")) ) {
      if( tclvalue(nor.s) != "" )
        comm = paste("pnorm(q=", tclvalue(valorsup), ", mean=",
                     tclvalue(nor.m), ", sd=", tclvalue(nor.s),
                     ") - ",
                     "pnorm(q=", tclvalue(valorinf), ", mean=",
                     tclvalue(nor.m), ", sd=", tclvalue(nor.s),
                     ")", sep="")
      if( tclvalue(nor.v) != "" )
        comm = paste("pnorm(q=", tclvalue(valorsup), ", mean=",
                     tclvalue(nor.m), ", sd=sqrt(", tclvalue(nor.v),
                     ")) - ",
                     "pnorm(q=", tclvalue(valorinf), ", mean=",
                     tclvalue(nor.m), ", sd=sqrt(", tclvalue(nor.v),
                     "))", sep="")
    }
    # esta función toma el comando R "fabricado",
    # lo "ejecuta" en R Commander y lo "imprime"
    # para que el usuario pueda ver el código
    doItAndPrint(comm)
    # activa la ventana de R Commander
    tkfocus(CommanderWindow())
  }
  # comportamiento si se pulsa el botón de ayuda
  OKCancelHelp(helpSubject="Distributions", reset="prob2")

    # diseño de la caja con texto y entradas de valores
  # de los parámetros
  # pestaña para la probabilidad
  probFrame <- tkframe(dataTab)
  tkgrid(probFrame, sticky = "nw")
  tkgrid(probFrame,
         tklabel(top, text="P("), valorinf.en,
```

```
        tklabel(top, text="< X \u2264"), valorsup.en,
        tklabel(top, text=")"),
        tklabel(top, text=""), tklabel(top, text=""),
        sticky="e")
  modelFrame <- tkframe(optionsTab)
  tkgrid(modelFrame, sticky = "nw")
  tkgrid(modelFrame,
        tklabel(top, text="BINOMIAL: Intentos"), bin.n.en, sticky="e")
  tkgrid(modelFrame,
        tklabel(top, text="Prob. de \u00e9xito"), bin.p.en, sticky="e")
  tkgrid(modelFrame,
        tklabel(top, text=""))
  tkgrid(modelFrame,
        tklabel(top, text="POISSON de media"), poi.m.en, sticky="e")
  tkgrid(modelFrame,
        tklabel(top, text=""))
  tkgrid(modelFrame,
        tklabel(top, text="EXPONENCIAL de media"), exp.m.en, sticky="e")
  tkgrid(modelFrame,
        tklabel(top, text="o bien de un proceso de intensidad"), exp.r.en,
                sticky="e")
  tkgrid(modelFrame,
        tklabel(top, text=""))
  tkgrid(modelFrame,
        tklabel(top, text="UNIFORME en el intervalo"), uni.a.en, sticky="e")
  tkgrid(modelFrame,
        tklabel(top, text=""), uni.b.en, sticky="e")
  tkgrid(modelFrame,
        tklabel(top, text=""))
  tkgrid(modelFrame,
        tklabel(top, text="NORMAL de media"), nor.m.en, sticky="e")
  tkgrid(modelFrame,
        tklabel(top, text="y desviaci\u00f3n t\u00edpica"), nor.s.en,
                sticky="e")
  tkgrid(modelFrame,
        tklabel(top, text="o varianza"), nor.v.en, sticky="e")
#  tkgrid(modelFrame, tklabel(top, text=""), tklabel(top, text=""),
#         buttonsFrame, sticky="w")
  tkgrid(modelFrame, buttonsFrame, sticky="e", columnspan=2)
```

```
  tkgrid.configure(valorinf.en, sticky="w")
  tkgrid.configure(valorsup.en, sticky="w")
  tkgrid.configure(bin.n.en, sticky="w")
  tkgrid.configure(bin.p.en, sticky="w")
  tkgrid.configure(poi.m.en, sticky="w")
  tkgrid.configure(exp.m.en, sticky="w")
  tkgrid.configure(exp.r.en, sticky="w")
  tkgrid.configure(uni.a.en, sticky="w")
  tkgrid.configure(uni.b.en, sticky="w")
  tkgrid.configure(nor.m.en, sticky="w")
  tkgrid.configure(nor.s.en, sticky="w")
  tkgrid.configure(nor.v.en, sticky="w")
  # acaba de definir la ventana
  # y coloca el cursor en la entrada correspondiente
  # al extremo inferior del intervalo de probabilidad
  dialogSuffix(use.tabs=TRUE, grid.buttons=TRUE)
}
```